بسم الله الرحمن الرحيم

كتاب في لغة الاسمبلي



الفصل الأول

مقدمة

في هذا الفصل سنتناول موضوع المعالجات الدقيقة وبرمجتها وسيتم التركيز علي المعالجات المستخدمة في الأجهزة الشخصية Personal Computers وهي المعالجات المصنعة بواسطة شركة Intel والمعالجات المتوافقة معها. وقد تمت الاستعانة بمجموعة من المراجع التي تغطي هذا الموضوع ولكن تم اعتماد المرجع الأول و هو كتاب Assembly Language Programming and Organization of The IBM PC كمرجع أساسي تم اللجوء إليه بصورة أساسية في كتابة هذه المادة هذا بالإضافة إلي مجموعة المراجع الأخرى والتي تم توضيحها في نهاية الكتاب

الخلفية المطلوبة Background

يجب الإلمام جيدا بكيفية التعامل مع الأنظمة الرقمية المختلفة وبالذات النظام الثنائي والسداسي عشري وإجادة التعامل مع العمليات الحسابية المختلفة من جمع وطرح وضرب وقسمة للأرقام المختلفة في تلك الأنظمة. كذلك يجب التعرف علي إحدى لغات البرمجة العليا علي الأقل ويفضل أن تكون إحدى اللغات التي تستعمل الهيكلة Structured Programming Language مثل الباسكال والسي ولكن يمكن بسهولة فهم البرامج بمجرد الإلمام بأي من لغات البرمجة العليا الأخرى. والهدف من ذلك هو كتابة بعض البرامج من خلال استعراض لغة التجميع ويفضل أن تكون لدينا بعض مهارات البرمجة المختلفة.

أسلوب تدريس المادة

سيتم التدريس باستخدام هذه المادة بالإضافة إلي مجموعة من برامج الكمبيوتر المصاحبة. ويتم ذلك عن طريق تدريس محاضرة واحدة أسبوعيا بواقع ساعتين للمحاضرة الواحدة، بالإضافة إلي ثلاثة ساعات عمليه يقوم

فيها الطالب بكتابة البرامج المطلوبة في نهاية كل مرحلة. يتم استلام البرامج أسبوعيا وتقييمها بواسطة الأستاذ ويتم ذلك باستخدام شبكة الحاسوب بالقسم. كما يتم عمل مجموعة من الاختبارات علي مدار فترة تدريس المادة هذا بالإضافة إلي الامتحان النهائي في نهاية الفترة المقررة.

محتويات المادة

تم تقسيم المادة لمجموعة من الفصول، كل فصل يمثل وحدة مستقلة ويجب دراسة الفصول بالترتيب حيث ان كل فصل يعتمد عادة علي الفصل السابق له. ويفضل الإجابة عن كل الأسئلة التي تأتي في نهاية كل فصل كما سيتم طلب كتابة مجموعة من البرامج في نهاية كل فصل. وتتمثل الفصول في الآتي:

الفصل الثاني : يتناول المعالجات الدقيقة بصورة عامة والمعالجات المنتجة بواسطة شركة Intel بصورة خاصة ثم يتعرض للتركيب الداخلي للمعالج 8088 والمسجلات المختلفة به وطريقة التخاطب مع الذاكرة.

الفصل الثالث: يوضح الشكل العام للأوامر في لغة التجميع وتعريف المتغيرات والثوابت بالإضافة إلي التعرف علي مجموعة من الأوامر الأساسية والتعرف علي الشكل

العام للبرنامج واستخدام نداءات المقاطعة للقيام بعمليات الإدخال والإخراج. في نهاية الفصل يتم كتابة برامج صغيرة وتجربتها.

الفصل الرابع: يتم فيه التعرف علي مسجل البيارق Flag Register وتأثر البيارق بالعمليات المختلفة وتوضيح حالات الفيضان المختلفة التي قد تحدث بعد تنفيذ عملية محددة.

الفصل الخامس: يتم فيه توضيح أوامر التفرع المختلفة وبعدها يتم التعرف علي كيفية تحويل البرامج الصغيرة من البرامج ذات المستوي العالي High Level Language ويتضمن ذلك تحويل أوامر التفرع والتكرار المختلفة إلى لغة التجميع. بعد ذلك تتم كتابة أحد البرامج الكبيرة نسبياً وتوضيح كيفية تحليل البرنامج إلى مرحلة الكتابة للبرنامج

الفصل السادس: يتناول أوامر الحساب والمنطق المختلفة وطريقة استخدامها في التعامل مع المسجلات ويتضمن ذلك أوامر الإزاحة والدوران. في نهاية الفصل تتم كتابة مجموعة من الإجراءات الفرعية لقراءة وكتابة الأرقام في النظامين الثنائي والسداسي عشري.

الفصل السابع: يتناول الحديث بالتفصيل عن المكدس Stack وكيفية التعامل معه، بعد ذلك يتم التعرف علي طريقة كتابة البرامج الفرعية

الفصل الثامن: يتم فيه التعرف علي أوامر الضرب والقسمة واستخدام البرامج الفرعية عن طريق كتابتها في ملف مختلف. ويتم كتابة برامج فرعية تقوم بقراءة أرقام عشرية من لوحة المفاتيح وطباعتها في الشاشة.

الفصل التاسع: يتم فيه التعرف علي أنماط العنونة المختلفة والمستخدمة في لغة التجميع كما يتم التعرف علي طريقة التعامل مع المصفوفات المختلفة.

الفصل العاشر: يتم فيه التعرف علي أوامر التعامل مع النصوص وسلاسل الحروف Strings.

الفصل الحادي عشر: يتم فيه استعراض مجموعة من البرامج التي تتعامل مع نظام التشغيل في أداء بعض الوظائف المحددة وذلك عن طريق ممارسة ما تم دراسته خلال هذا المقرر وربط ذلك ببعض الأمثلة العملية المهمة.

الهدف من المادة

في كثير من الأحيان نضطر لكتابة بعض البرامج الخاصة جداً والتي تتعامل مع مكونات النظام من أجهزة مختلفة وعند الانتهاء من دراسة هذه المادة يكون الطالب قد تعرف علي كيفية التعامل مع المعالج الدقيق مباشرة ومعرفة ما يدور في المستوي الأدنى للجهاز Low_Level ويصبح قادراً علي كتابة برامج تتعامل مع النظام في أدق تفاصيله كما يصبح بإمكانه تحليل وفهم أي برنامج كتب بلغة التجميع. ويصبح الطالب جاهزاً لدراسة مادة برمجة النظم Systems Programming.

الفصل الثاني

المعالجات وتنظيم الحاسب الشخصي

مقدمة:

تعتمد الأجهزة المتوافقة مع نظام IBM علي المعالجات من عائلة المعالج Intel. في هذا الفصل سيتم عرض عام للمعالجات من عائلة المعالج 8086 في الجزء الأول حيث يتم التعرف علي المعالج 8086 مع توضيح المسجلات المختلفة و

استخدامات كل مسجل ثم يتم توضيح عملية تقسيم الذاكرة إلي قطاعات Segments.

عائلة المعالجات Intel 8086

تعتمد الحاسبات الشخصية المتوافقة مع IBM علي المعالجات من النوع Intel وهي تشمل المعالجات 8086 و 8088 و 80286 و 80386 و 80486 و أخيراً المعالج Pentium حيث يتم استخدام المعالج لبناء نظام حاسوب بخصائص محددة كما في حالات استخدام المعالج 8088 لبناء الحاسوب من النوع IBM PC و استخدام المعالج 80286 لبناء الحاسوب المسمى (XT (eXtended Technology كما تم بناء النظام (AT (Advanced Technology مع ظهور المعالج 80386.

ثم بعد ذلك ونتيجة لأهمية وضع نظم ثابتة ومعروفة للجميع ظهرت أنظمة (.ISA (Industry Standard Arch و EISA (Extended ISA) وهي أنظمة تستعمل المعالجين 80386 و 80486 .

مع ظهور المعالج الجديد والمسمي Pentium ظهرت الحاجة لأنظمة جديدة ذات سرعة عالية فظهرت أنظمة الناقل المحلي

Local Bus Systems مثل نظام PCI ونظام VESA وذلك للاستفادة من الإمكانات الجديدة للمعالج.

مما يجدر ذكره أن المعالجات من عائلة Intel حافظت علي التوافقية في تصميم المعالجات بحيث يتم استيعاب وتنفيذ البرامج التي تمت كتابتها لتعمل مع المعالجات القديمة في المعالجات الجديدة بدون مشاكل وهو ما يسمى بتوافقية البرامج Software Compatibility وهي ميزة كبيرة في التصميم حيث تم الاحتفاظ بالبرامج القديمة دون أي تعديل مع إمكانية تشغيل البرامج الجديدة ذات الإمكانات الجديدة والتي لم تكن موجودة في المعالجات القديمة. فيما يلي سنتناول المعالجات المختلفة بشيء من التفصيل وذلك بتوضيح الخصائص العامة للمعالج من حيث طول الكلمة Word Length وأقصي قيمة للذاكرة بالإضافة لبعض الخصائص العامة.

المعالج 8086 والمعالج 8088

قامت شركة Intel في عام 1978 بطرح المعالج 8086 وهو معالج يتعامل مع كلمة بطول 16-bits (يتم التعامل 16-bitفي المرة الواحدة).بعد ذلك وفي سنة 1979 تم طرح المعالج 8088 وهو مشابه للمعالج 8086 من

ناحية التركيب الداخلي ولكنه مختلف عنه في التعامل العام الخارجي حيث يتم فيه التعامل الخارجي بكلمه طولها 8-bits بينما يتعامل المعالج 8086 باستخدام نبضة سريعة وبالتالي فان أداءه افضل (زيادة سرعة النبضة تعنى زيادة التردد وبالتالي نقصان الزمن اللازم لتنفيذ أمر محدد ويتم تعريف سرعة المعالج بتحديد التردد الأقصى الذي يعمل به وتقاس وحدة التردد بالميجاهيرتز MHz).

قامت شركة IBM باختيار المعالج 8088 لبناء الحاسب الشخصي IBM PC وذلك لسهولة التعامل معه بالإضافة إلي رخص التكلفة حيث كان من المكلف في ذلك الوقت بناء الحاسب على المعالج 8086 ذات الـ16-bit وذلك بسبب ارتفاع تكلفة بناء نظام بوحدات مساعده تتعامل مع كلمة بطول 16-bit في ذلك الزمن.

يتعامل المعالجان 8086 و8088 بنفس التعليمات وهما يمثلان نقطة البداية التي بدأت منها المعالجات الجديدة والتي يتم استعمالها في أجهزة الحاسب الشخصية وبالتالي فان البرامج التي تعمل على المعالجين 8086 و 8088 مازالت صالحة للعمل في المعالجات الجديدة وهو ما أسميناه بالتوافقية في البرامج.

المعالجان 80186 و 80188

يعتبر المعالجان 80186 و 80188 تطويراً للمعالجين 8086 و 8088 وذلك عن طريق تنفيذ كل التعليمات التي كانت مستخدمة في المعالجات القديمة بالإضافة إلي بعض الأوامر المختصة بالتعامل مع بعض الوحدات المساعدة Support Chips. كذلك تمت إضافة بعض الأوامر الجديدة وهى ما تسمى بال Extended Instruction . وعموماً لـم يتم استعمال المعالجين في الأجهزة بصورة كبيرة وذلك نسبة لعدم وجود فارق كبير عن سابقيهما بالإضافة إلى ظهور المعالج الجديد 80286 في الأسواق.

المعالج 80286 :-

تم طرح المعالج 80286 في سنة 1982 م وهو معالج يتعامل مع كلمة بطول 16 Bits ولكنه أسرع بكثير من المعالج 8086 حيث تصل سرعته إلي 12.5 MHZ

وذلك مقارنة مع 10 MHZ للمعالج 8086. كذلك تميز المعالج 80286 بالمزايا التالية :-

1 - نمطين للأداء Two Modes Of Operations

المعالج 80286 يمكنه العمل في نمطين وهما النمط الحقيقي Real Mode والنمط المحمى Protected Mode .

في النمط الحقيقي يعمل المعالج 80286 كمعالج من النوع 8086 وبالتالي فان البرامج التي تمت كتابتها للمعالج 8086 تعمل في هذا النمط بدون أي تعديل.

أما في النمط المحمى فانه يمكن أن يتم تشغيل أكثر من برنامج في وقت واحد Multi_Tasking وبالتالي يلزم حماية كل برنامج من التعديل بواسطة برنامج آخر يعمل في الذاكرة في نفس الوقت وذلك بتخصيص منطقة محددة من الذاكرة لكل برنامج على حدة ومنع البرنامج من التعامل مع مناطق الذاكرة التي تخص البرنامج الآخر.

2 - ذاكرة أكبر :-

يمكن للمعالج 80286 التخاطب مع ذاكرة تصل

إلي 16 MByte وذلك في النمط المحمى (مقابل

1 MBYTE للمعالج 8086) .

3 - التعامل مع الذاكرة الافتراضية :-

حيث يتم ذلك في النمط المحمى وذلك بإتاحة

الفرصة للمعالج للتعامل مع وحدات التخزين

الخارجية لتنفيذ برامج كبيرة تصل لـ 1 GBYTE

(لاحظ أن أقصى قيمة للذاكرة هي 16 MBYTE

فقط) وسيتم التحدث عن هذه الطريقة بالتفصيل

في مادة نظم التشغيل .

المعالج 80386 :-

في عام 1985 تم إنتاج أول معالج يتعامل مع كلمة

بطول 32 BITS وهو المعالج 80386 وهو أسرع

بكثير من المعالج 80286 وذلك لمضاعفة طول

الكلمة (من 16_BIT إلى32_BIT) ونسبة للسرعة

الكبيرة التي يتعامل بها المعالج والتي تصل إلي 40

MHZ فإنه يقوم بتنفيذ عدد كبير من الأوامر في

عدد أقل من عدد النبضات التي يستغرقها المعالج

80286 .

يستطيع المعالج 80386 التعامل مع النمط الحقيقي والنمط المحمى حيث يعمل في النمط الحقيقي كالمعالج 80386 وفى النمط المحمى كالمعالج 80286. ذلك بالإضافة إلي نمط جديد يسمى بالنمط الافتراضي للمعالج 8086 (VIRTUAL 8086 MODE) وهو نمط مصمم لجعل أكثر من برنامج من برامج المعالج 8086 تعمل في الذاكرة في وقت واحد .

يستطيع المعالج 80386 التعامل مع ذاكرة يصل حجمها إلي 4 Gbytes وذاكرة افتراضية يصل حجمها إلى 64 T BYTES.

توجد كذلك نسخة رخيصة من المعالج تسمى 80386SX وهى تحتوى على نفس الشكل الداخلي للمعالج 80386 ولكنها خارجيا تتعامل مع 16 BITS .

المعالج 80486 :-

في عام 1989 ظهر المعالج 80486 وهو عبارة عن نسخة سريعة من المعالج 80386 حيث يحتوى على كل مزايا المعالج 80386 بالإضافة للسرعة الكبيرة وتنفيذ الكثير من الأوامر المستخدمة بكثرة في

نبضة واحدة فقط كذلك احتوائه علي المعالج المساعد 80387 والمختص بالعمليات الحسابية التي تحتوى على أعداد حقيقية حيث كانت هذه العمليات تستغرق وقتاً طويلاً من المعالج 80386 مما تطلب وجود المعالج 80387 والذي يسمي بالمعالج المساعد الرياضي Math. Co_Processor وقد تم دمج هذا المعالج مع المعالج 80386 بالإضافة إلي ذاكرة صغيرة تسمي بالـ Cache Memory (وهي ذاكرة ذات زمن وصول صغير جداً ويتم استخدامها كوسيلة لتبادل البيانات بين الذاكرة العادية والمعالج الدقيق)وحجمها 8 Kbytes.

يعتبر المعالج 80486 أسرع من المعالج 80386 والذي يعمل علي نفس التردد بحوالي ثلاث مرات. هذا بالإضافة إلي أن المعالج 80486 يعمل علي ترددات (سرعات) عالية جداً تصل إلي 100 M Hz. أما المعالج 80486SX فهو كالمعالج 80486 تماماً من حيث العمل الداخلي فيما عدا أنه لا يحتوي علي معالج رياضي داخله. وقد ظهرت عدة إصدارات من المعالج 80486 ولكن لا توجد اختلافات جوهرية كبيرة بينها والمجال هنا لا يتسع لذكرها.

المعالج Pentium

المعالج Pentium هو آخر إصدارات شركة Intel وهو أول معالج يتعامل مع كلمة بطول 64 Bits بالإضافة إلي السرعة العالية جداً التي يعمل بها مقارنة بالمعالج 80486 هذا بالإضافة إلي زيادة حجم الذاكرة الداخلية Cache Memory.
وقد ظهرت إصدارات مختلفة للمعالج Pentium ازدادت فيها سرعة المعالج وتمت إضافة إمكانات إضافية إليه فيها مثل MMX والذي يمتاز بأن به أوامر للتعامل مع الوسائط المتعددة.

التركيب الداخلي للمعالج 8088 والمعالج 8086

في هذا الجزء سيتم التعرف علي التركيب الداخلي للمعالج وذلك عن طريق التعرف علي المسجلات المختلفة الموجودة داخل المعالج ووظيفة كل مسجل وسيتم في الأجزاء التالية مناقشة الأوامر المختلفة التي يتم استخدامها في التعامل مع المعالج. ونسبة لتوافقية البرامج التي تم الحفاظ عليها في المعالجات الجديدة سنجد أن هذه التعليمات يمكن استخدامها مع المعالجات الحديثة وحتى الـ Pentium .

المسجلات

يتم تخزين البيانات داخل المعالج في المسجلات، ويتم تقسيم المسجلات إلي مسجلات بيانات ويتم فيها التعامل مع البيانات من حيث التخزين وإجراء العمليات الحسابية والمنطقية ومسجلات عناوين ويتم فيها تخزين العناوين المختلفة ومسجل الحالات وهو يحتوي علي حالة المعالج بعد تنفيذ أمر محدد. ويحتوي المعالج علي عدد 14 مسجل وسنقوم في الجزء التالي بتوضيح أسماء ووظيفة كل مسجل.

مسجلات البيانات DX,CX,BX,AX

يتم استخدام هذه المسجلات الأربعة في التعامل مع البيانات داخل المعالج و يمكن للمبرمج التعامل مباشرة مع هذه المسجلات. وبالرغم من أن المعالج يستطيع أن يتعامل مع بيانات فى الذاكرة إلا أن التعامل مع المسجلات يكون أسرع بكثير من التعامل مع الذاكرة (يلزمه عدد اقل من النبضات) و بالتالي نفضل دائماً التعامل مع المسجلات لسرعتها . وهذا سبب زيادة عدد المسجلات في المعالجات الحديثة.

يمكن التعامل مع كل من هذه المسجلات على انه وحده واحدة بحجم 16-BITS أو على وحدتين كل واحدة

بسعة 8-BITS إحداهما العليا HIGH و الثانية المنخفضة LOW مثلا يمكن التعامل مع المسجل AX على انه مسجل بحجم 16-BITS أو التعامل مع النصف العلوي (H IGH AH(على انه مسجل 8-BITS و المسجل المنخفض AL (LOW) على انه مسجل8-BITS. وبالمثل مع المسجلات D,C,B و بالتالي يصبح لدينا 8 مسجلات من النوع -8 BITS أو أربعة مسجلات من النوع 16-BITS.

بالرغم أن المسجلات الأربعة ذات استخدامات عامه GENERAL PURPOSE REGISTERS بحيث يمكن استخدامها في أي استخدامات عامه إلا أن لكل مسجل استخداماً خاصاً نتناوله في الجزء التالي :-

1-المسجل AX(Accumulator)

يعتبر المسجل AX هو المسجل المفضل للاستخدام في عمليات الحساب و المنطق و نقل البيانات و التعامل مع الذاكرة و موانئ الإدخال و الإخراج. و استخدامه يولد برامج اقصر ويزيد من كفاءة البرنامج. حيث يجب مثلا في عمليه ضرب رقمين وضع أحد الرقمين فيه مع وضع القيمة المطلوب

إخراجها إلي ميناء خروج محدد فيه ثم تتم قراءه

القيمة التي يتم إدخالها من ميناء خروج محدد فيه

دائما. وعموما يتم التعامل مع المسجلAX على انه

أهم المسجلات الموجودة في المعالج.

2-المسجل BX (Base Register)

يستخدم المسجل BX في عنونه الذاكرة حيث

تتطلب بعض العمليات التعامل مع الذاكرة

بمؤشر محدد ويتم تغيير قيمه المؤشر لإجراء عمليه

مسح لجزء محدد من الذاكرة كما سنري فيما بعد.

3-المسجل CX (Count Register)

يتم استخدام المسجل CX كعداد للتحكم بعدد

مرات تكرار مجموعه محدده من التعليمات.كذلك

يتم استخدامه في تكرار عمليه دوران مسجل

لعدد محدد من المرات.

4-المسجل DX (Data Register)

يتم استخدامه في عمليات الضرب والقسمة كذلك

يتم استخدامه كمؤشر لموانئ الإدخال والإخراج

عند استخدام عمليات الإدخال والإخراج.

مسجلات المقاطع CS, DS, SS, ES

يتم استخدام هذه المسجلات لتحديد عنوان محدد في الذاكرة. ولتوضيح وظيفة هذه المسجلات يجب في البداية توضيح طريقة تنظيم الذاكرة .

نعلم أن المعالج 8088 يتعامل مع 20 إشارة عناوين (ناقل العناوين Address Bus يحتوي على 20 إشارة) وبالتالي يمكن مخاطبة ذاكرة تصل إلي 2^{20} = 1,048,576 أي 1 Mbytes .

ونجد أن عناوين أول 5 خانات في الذاكرة هي :

00000 h = 0000 0000 0000 0000 0000
00001 h = 0000 0000 0000 0000 0001
00002 h = 0000 0000 0000 0000 0010
00003 h = 0000 0000 0000 0000 0011
00004 h = 0000 0000 0000 0000 0100

ولأن العناوين في الصورة الثنائية تكون طويلة جداً فمن الأسهل التعامل مع العناوين بكتابتها في الصورة السداسية عشر وبالتالي يكون عنوان أول خانة في الذاكرة هو 00000h وعنوان آخر خانة هو FFFFFh .

مما سبق يتضح أن العنوان يتكون من 20 خانة بينما كل المسجلات الموجودة داخل المعالج ذات طول مقداره 16 خانة فقط مما يجعل مخاطبة الذاكرة كلها مستحيلة باستخدام مسجل واحد فقط (لاحظ أن المسجل الواحد باستطاعته مخاطبة ذاكرة تصل إلي 64 Kbytes فقط) ونتيجة لظهور هذه المشكلة تم تقسيم الذاكرة إلي مجموعة من المقاطع Segments كل مقطع بسعة 64 K Bytes كما سنوضح في الجزء التالي .

مقاطع الذاكرة

مقطع الذاكرة هو جزء متصل بطول 2^{16} = 64 Kbytes وكل مقطع في الذاكرة يتم تحديده برقم محدد يسمي رقم المقطع Segment Number وهو رقم يبدأ بالرقم 0000h وينتهي بالرقم FFFFh.

بداخل المقطع يتم تحديد العنوان بواسطة إزاحة محددة Offset وهذه الإزاحة عبارة عن بُعد الموقع المحدد من بداية المقطع وهو رقم بطول 16 Bytes أي تتراوح قيمته بين الرقمين 0000h و FFFFh.

وبالتالي لتحديد عنوان محدد في الذاكرة يجب توضيع قيمة كل من المقطع والإزاحة وبالتالي تتم كتابة العنوان علي الصورة :

Segment : Offset

وهو ما يسمي بالعنوان المنطقي Logical Address فمثلاً العنوان AABB:5566 يعني الإزاحة 5566 داخل المقطع AABB.

للحصول علي العنوان الفيزيائي يتم ضرب قيمة المقطع في الرقم 16 (إزاحته لليسار بمقدار أربعة خانات ثنائية أو خانة واحدة سداسية عشر) ويتم بعد ذلك إضافة قيمة الإزاحة إليه وبالتالي فإن العنوان الفيزيائي المناظر للعنوان AABB:5566 هو

A ABB0

+ 5566

= B 1116 (العنوان الفيزيائي (بطول 20 خانة)

وبالتالي يصبح العنوان الفيزيائي = رقم المقطع * 16 + قيمة الإزاحة

مواضع المقاطع LOCATIONS OF SEGMENTS

يتضح مما سبق أن المقطع الأول في الذاكرة يبدأ بالعنوان 0000:0000 أي 00000 وآخر عنوان داخل المقطع هو العنوان 0000:FFFF أي العنوان 0FFFF بينما يبدأ المقطع الثاني في العنوان 0001:0000 أي العنوان 00010 وينتهي بالعنوان 0001:FFFF أي العنوان 1000F. وكما نرى فان هناك كثيراً من التداخل في المقاطع داخل الذاكرة . الشكل(1) يوضح الذاكرة وعناوين المقاطع المختلفة بداخلها

محتويات الذاكرة	العنوان	
45	1001F	نهاية المقطع رقم 2
	....	
45	1000F	نهاية المقطع رقم 1
	....	
53	0FFFF	نهاية المقطع رقم 0
	....	

29	00020	بداية المقطع رقم 2
76	00010	بداية المقطع رقم 1
54	00000	بداية المقطع رقم 0

الشكل (1)

في الشكل(1) يتضح أن المقطع يبدأ بعد كل 16 خانة في الذاكرة . وعلى ذلك تسمى كل 16 خانة في الذاكرة بفقرة Paragraph. ويسمى أي من العناوين التي تقبل القسمة على العدد 10h بحدود الفقرات Paragraph Boundaries .

ولأن هنالك تداخلاً في القطاع فان تحديد العنوان الفيزيائي قد يتم بأكثر من طريقة أي عن طريق اكثر من تشكيلة في عنوان المقطع وعنوان الإزاحة . والأمثلة التالية توضح ذلك :

مثال :- قم بتحديد قيمة الإزاحة المطلوبة لتحديد العنوان 1256A وذلك في :

أ- القطاع 1256 ب- القطاع 1240

الحل :

يتم استعمال المعادلة : العنوان = المقطع * 16 +
الإزاحة

أ- افترض أن قيمة الإزاحة المطلوبة X بالتعويض
في المعادلة نجد أن

$$1256A = 1256*10h+X$$

$$1256A = 12560+X$$

$$000A = X$$

وبالتالي فان العنوان هو 1256:000A

ب – بإتباع نفس الطريقة التي اتبعناها في الجزء
السابق

افترض أن قيمة الإزاحة المطلوبة X بالتعويض في
المعادلة نجد أن

$$1256A = 1240*10h+X$$

$$1256A = 12400+X$$

$$016A = X$$

وبالتالي فان العنوان هو 1240:016A

أي أن العنوانين يشيران إلي نفس العنوان في الذاكرة

1256A = 1256:000A = 1240:016A

من الممكن أيضاً معرفة رقم المقطع بمعرفة العنوان الفيزيائي وقيمة الإزاحة كما في المثال التالي :

مثال

ما هو عنوان المقطع لتحديد العنوان 80FD2h إذا كانت الإزاحة تساوي 8FD2h

باستعمال المعادلة : العنوان = المقطع * 16 + الإزاحة،

نجد أن

80FD2h = قيمة مسجل المقطع * 10h + BFD2h

قيمة مسجل المقطع = 7500h

بعد توضيح عملية تقسيم الذاكرة لمقاطع مختلفة يمكننا الآن شرح عمل مسجلات المقاطع المختلفة، حيث يتكون البرنامج من مجموعة من الأوامر بالإضافة إلي مجموعه من المتغيرات هذا بالإضافة إلي الحاجة لاستخدام مكدس البيانات Stack والذي سنوضح طريقة استخدامه وعمله لاحقاً.

يتم وضع البرنامج في مقطع البرنامج Code Segment ووضع البيانات في مقطع البيانات Data Segment وكذلك

المكدس حيث له مقطع المكدس Stack Segment ولدينا مقطع إضافي يسمي بالـ Extra Segment.

مسجل مقطع البرنامج Code Segment Register (CS)

يحتوي هذا المسجل علي عنوان مقطع البرنامج Code Segment Address حيث يتم تحديد مقطع محدد في الذاكرة يتم وضع البرنامج فيه، بعد ذلك يلزم تعريف ذلك العنوان للمعالج حيث سيتم تنفيذ البرنامج؛ لذلك يجب تحديد عنوان هذا المقطع ووضعه في مسجل خاص يسمي بمسجل مقطع البيانات Code Segment Register (CS) ويتم تحديد قيمة الإزاحة باستخدام مسجل مؤشر التعليمات Instruction Pointer والذي سيتم التحدث عنه لاحقاً.

مسجل مقطع البيانات Data segment Register (DS)

يحتوي هذا المسجل علي عنوان مقطع البيانات Data Segment Address حيث يتم تعريف البيانات التي يتعامل معها البرنامج في منطقة محددة من الذاكرة (وتسمي مقطع البيانات) ويتم تحديد عنوان هذا المقطع ووضعه في المسجل DS. بعد ذلك يمكن مخاطبة

الذاكرة والتعامل مع المتغيرات المختلفة باستخدام مسجلات أخري تحوي قيمة الإزاحة المطلوبة.

مسجل مقطع المكدس Stack Segment Register (SS)

يتم تحديد جزء من الذاكرة والتعامل معه كمكدس حيث يعمل المكدس بطريقة Last In First Out) (LIFO)ويتم استعماله في مجموعة من العمليات أهمها عملية النداء لبرامج فرعية كما سنري لاحقاً ويتم استعمال مجموعة المسجلات لتحوي قيمة الإزاحة ومن أهمها مؤشر المكدس Stack Pointer (SP) .

مسجل المقطع الإضافي Extra Segment Register (ES)

ويتم استخدام هذا المسجل لتحديد ومخاطبة مقطع إضافي حيث تلزم في بعض الأحيان عملية مخاطبة أكثر من مقطع في وقت واحد (مثل نقل كمية من البيانات في الذاكرة من مكان محدد لمكان آخر في مقطع بعيد وبالتالي لا يكفي مسجل البيانات فقط ولكن نحتاج لمسجل إضافي لتحديد المقطع الآخر فيتم استعمال المقطع الإضافي ES)

مسجلات المؤشرات والفهرسة Index and Pointer Registers (SP, BP, SI, DI)

يتم استخدام هذه المسجلات مع مسجلات المقاطع التي تحدثنا عنها في الجزء السابق للتخاطب مع عناوين محددة في الذاكرة، وعكس مسجلات المقاطع يمكن إجراء عمليات الحساب والمنطق علي هذه المسجلات.

مؤشر المكدس Stack Pointer (SP)

يتم استخدام هذا المسجل مع مقطع المكدس وسيتم التحدث بالتفصيل عن المكدس في الفصول القادمة.

مؤشر القاعدة Base Pointer (BP)

يتم استخدام هذا المسجل أساساً للتخاطب مع البيانات الموجودة في المكدس ولكنه عكس مؤشر المكدس حيث يمكن استخدامه لمخاطبة الذاكرة في مقاطع أخري غير مقطع المكدس.

مسجل فهرسة المصدر Source Index (SI)

يستخدم هذا المسجل في مخاطبة الذاكرة في مقطع البيانات حيث يقوم بالإشارة إلي بداية (أو نهاية) منطقة محددة من الذاكرة مطلوب التعامل معها؛

وبتغيير قيمة هذا المسجل في كل مرة يتم التعامل مع كل هذه المنطقة من الذاكرة.

مسجل فهرسة المستودع Destination Index (DI)

هذا المسجل يستخدم مثل مسجل فهرسة المصدر SI حيث يشير هذا المسجل إلي عنوان الذاكرة الذي سيتم تخزين البيانات فيه ويتم ذلك عادة باستخدام المقطع الإضافي ES وهناك مجموعة من الأوامر التي تتعامل مع النصوص والتي تفترض أن عنوان المصدر وعنوان المستودع يتم تحديدهما في هذين المسجلين.

مؤشر التعليمات أو الأوامر Instruction Pointer (IP)

كل المسجلات التي تحدثنا عنها حتى الآن يتم استخدامها في مخاطبة البيانات المخزنة في الذاكرة. لمخاطبة البرنامج يلزم المعالج معرفة عنوان أول أمر في البرنامج المطلوب تنفيذه، بعد ذلك يقوم المعالج بتحديد عنوان الأمر التالي ويستمر في تنفيذ البرنامج.

يتم تخزين الإزاحة للأمر المطلوب تنفيذه في مؤشر التعليمات أو الأوامر Instruction Pointer (IP) حيث يتم ذلك في مقطع البرنامج Code Segment وبالتالي فإن عنوان الأمر المطلوب تنفيذه هو CS:IP. ولا

يمكن مخاطبة مؤشر التعليمات مباشرة من داخل البرنامج وإنما يتم تغيير قيمته بطريقة غير مباشرة مثل حالات التفرع إلي عنوان محدد حيث يتم وضع قيمة ذلك العنوان في مؤشر التعليمات وذلك في حالة حدوث عملية التفرع.

مسجل البيارق Flags Register

يحتوي هذا المسجل علي مجموعة من البيارق (الأعلام) وهي نوعان: بيارق الحالة وبيارق التحكم. بالنسبة لبيارق الحالة فهي توضح حالة المعالج بعد تنفيذ كل عملية لتوضيح حالة النتيجة حيث يمكن عن طريق هذه البيارق معرفة النتيجة (مثلاً إذا كان بيرق الصفر قد تم رفعه فمعني ذلك أن نتيجة آخر عملية تساوي صفر) وبالتالي يمكن اختبار البيارق المناسبة واتخاذ القرارات المناسبة. أما بيارق التحكم فيتم استعمالها لإخطار المعالج بالقيام بشيء محدد مثلاً يمكن استخدام بيرق المقاطعة Interrupt Flag ووضع القيمة صفر فيه وبالتالي فإننا نطلب من المعالج أن يتجاهل نداءات

المقاطعة الواردة إليه من لوحة المفاتيح مثلاً (أي لا يتم استقبال مدخلات من لوحة المفاتيح) وسيتم التحدث عن هذه البيارق بالتفصيل لاحقاً.

تنظيم الذاكرة في الحاسب الشخصي Memory Organization

يتعامل المعالج 8088 مع ذاكرة بطول 1Mbyte. ولا يمكن استخدام كل الذاكرة في البرامج التي يتم كتابتها ولكن هناك مناطق في الذاكرة محجوزة لأغراض محددة فمثلاً لدينا الجزء الأول من الذاكرة بطول 1KByte محجوز لعناوين نداءات المقاطعةInterrupt Vector Table كذلك هناك أجزاء مخصصة لبرامج النظام الأساسي للإدخال والإخراج BIOS والذي يقوم بعمليات الإدخال و الإخراج في الجهاز؛ و يتم تخزينه داخل ذاكرة قراءة فقط ROM(READ ONLY MEMORY) وهو الذي يقوم ببدء تشغيل الجهاز في المرحلة الأولي.

كذلك توجد منطقة في الذاكرة مخصصة لوحدة العرض(الشاشة VIDEO DISPLAY MEMORY).

موانئ الإدخال والإخراج I/O PORTS

يتعامل المعالج 8088 مع 64KB من عناوين الإدخال والإخراج وذلك للتعامل مع الأجزاء الإضافية والخارجية . وعموما لا يفضل التخاطب مع موانئ الإدخال والإخراج مباشرة إلا في بعض الحالات الخاصة وذلك بسبب احتمال تغير العناوين في بعض الأجهزة ويفضل أن يتم التعامل مع الأجهزة عن طريق نداءات لنظام التشغيل ليقوم هو بهذه المهمة .

تمارين

1- ما هو الفرق بين المعالج 80286 والمعالج 8088 ؟

2- ما هو الفرق بين المسجل والموقع المحدد في الذاكرة ؟

3- اذكر وظائف مسجلات البيانات DX,CX,BX,AX .

4- ما هو العنوان الفيزيائي للموقع المحدد بالعنوان 0A51:CD90 ؟

5- موقع في الذاكرة عنوانه 4A37B احسب:

أ- الإزاحة إذا كان عنوان القطاع هو 40FF .

ب- عنوان القطاع إذا كانت قيمة الإزاحة 123B.

6 - ما هي حدود الفقرات في الذاكرة ؟

الفصل الثالث

مدخل إلي لغة التجميع

بعد توضيح التركيب الداخلي للمعالج 8088 والتعرف على المسجلات المختلفة الموجودة به سنتناول في هذا الفصل كيفية كتابة وتجهيز وتشغيل برنامج لغة التجميع وبنهاية الفصل سنستطيع أن نكتب برنامج لغة تجميع وان نقوم بتشغيله ورؤية النتيجة.

كأي لغة سنبدأ بتوضيح الصيغة العامة للأوامر وهى صيغه بسيطة جداً في لغة التجميع. بعدها سنوضح طريقة تعريف المتغيرات داخل البرنامج وبعدها نستعرض بعض أوامر نقل البيانات وأوامر العمليات الحسابية البسيطة. في النهاية سنستعرض الشكل العام للبرنامج والذي ستلاحظ انه يتكون من جزء خاص بالأوامر وجزء ثاني خاص بالبيانات وجزء أخير خاص بالمكدس، سيتم استخدام بعض النداءات البسيطة لنظام التشغيل ليقوم بتنفيذ عمليات الإدخال والإخراج.

في النهاية سيتم توضيح كيفية تحويل برنامج لغة التجميع إلى لغة الآلة وتشغيل البرنامج في صورته النهائية.

تعليمات لغة التجميع :-

يتم تحويل برنامج لغة التجميع للغة الآلة بواسطة برنامج يسمى Assembler وبالتالي يجب كتابة التعليمات بصوره محدده

حتى يتعرف عليها الـ Assembler، وفى هذا الجزء سنتناول الشكل العام للأوامر المستخدمة.

يتكون البرنامج من مجموعه من التعليمات أو الأوامر بحيث يحتوى كل سطر على أمر واحد فقط كما أن هنالك نوعين من التعليمات.

الأوامر أو التعليمات Instructions والتي يقوم الـ Assembler بتحويلها إلي لغة الآلة والإيعازات -Assembler Directives وهى إيعاذات للـ Assembler للقيام ببعض العمليات المحددة مثل تخصيص جزء من الذاكرة لمتغير محدد وتوليد برنامج فرعى.

كل الأوامر في لغة التجميع تأخذ الصورة

NAME OPERATION OPERAND(S) COMMENT

- يتم الفصل بين الحقول بواسطة مفتاح الـTAB أو المسطرة (SPACE) أي يكون هناك فراغ واحد على الأقل بين كل حقل والحقل التالي.
- يتم استخدام الاسم NAME في حالة حدوث عملية تفريع لهذا الأمر (لهذا السطر من البرنامج) في جزء ما من البرنامج وهو حقل اختياري.
- الحقل Operation يحتوى على الأمر المطلوب تنفيذه.

- الحقل Operation(s) يحتوى على المعامل أو المعاملات المطلوب تنفيذها بواسطة الأمر المحدد ويعتمد على نوع الأمر. (لاحظ أن هناك بعض الأوامر لا تتطلب وجود هذا الحقل).
- حقل الملحوظات الـ Comments يستخدم عادة للتعليق على الأمر الحالي وهو يستخدم لتوثيق البرنامج.

كمثال للتعليمات

CX , 5 ; MOV Srart:
initialize counter

هذه الأمر ذو عنوان Start والأمر المستخدم MOV والمعاملات هي CX والرقم 5 ومعني ذلك هو وضع الرقم 5 في المسجل CX وحقل الملاحظات يوضح أن 5 هي القيمة الابتدائية للعداد.

ومثال للإيعازات

Proc Main

وهذا الإيعاز يقوم بتعريف برنامج فرعي (إجراء) باسم Main.

فيما يلي سنتحدث عن الحقول المختلفة بالتفصيل:

حقل العنوان Name Field

يتم استخدام هذا الحقل لإعطاء عنوان لأمر محدد أو لإعطاء اسم لبرنامج فرعي كذلك لإعلان أسماء

المتغيرات، يتم تحويل هذا الحقل إلي عناوين في الذاكرة.

يمكن أن يكون هذا الحقل بطول حتى 31 حرف وغير مسموح وجود مسافات بداخل الحقل كذلك لا يستخدم الحرف "." إلا في بداية الاسم ولا يبدأ برقم ولا يتم التفريق بين الحروف الكبيرة والصغيرة فيه.

أمثلة لأسماء مقبولة:

start – counter - @character – sum_of_digits - $1000 – done? - .test

أمثلة لأسماء غير مقبولة:

two words يحتوي علي فراغات

2abc يبدأ برقم

a45.ab يحتوي علي الحرف (.) في منتصفه

حقل التعليمة (الأمر) Operation Field

يحتوي هذا الحقل علي الأمر OpCode المطلوب تنفيذها في هذا السطر ويجب أن تكون إحدى التعليمات المعروفة للبرنامج الذي سيقوم بمعالجة البرنامج وهو الـ Assembler حيث سيقوم بتحويلها إلي

لغة الآلة كمثال لذلك التعليمات Sub و Add و Mov وكلها تعليمات معرفة وسيتم الحديث عنها بالتفصيل لاحقاً.

أما إذا كانت إيعازاً Pseudo-Op فلا يتم تحويلها للغة الآلة ولكنها لإخطار الـ Assembler ليقوم بشيء محدد مثلاً Proc تستخدم لتعريف برنامج فرعي Procedure

حقل المعاملات Operand Field

يحتوي هذا الحقل علي المعاملات من مسجلات ومتغيرات وثوابت والتي سيتم تنفيذ الأمر الحالي عليها (مثل عملية الجمع مثلاً) ويمكن لهذا الحقل أن يحتوي علي قيمتين أو قيمة واحدة أو لا يحتوي علي أي قيمة علي الإطلاق وذلك حسب نوع الأمر المستخدم والأمثلة التالية توضح ذلك

الأمر	المعاملات
NOP	لا توجد معاملات
INC CX	يوجد معامل واحد وهو المسجل CX
ADD Word1 , 2	يوجد معاملان وهما المتغير Word1 والرقم 2

في حالة الحقول ذات المعاملين يكون المعامل الأول هو الذي سيتم تخزين النتيجة فيه ويسمي بالمستودع destination Operand وهو يكون إما أحد المسجلات أو موقع محدد في الذاكرة (لاحظ أن بعض الأوامر لا تقوم بتخزين النتيجة أصلاً) أما المعامل الثاني فيحتوي علي المصدر Source Operand وعادة لا يتم تغيير قيمته بعد تنفيذ الأمر الحالي.

أما بالنسبة للإيعازات فيحتوي المعامل عادة علي معلومات إضافية عن الإيعاز.

حقل التعليقات والملاحظات Comment Field

يحتوي هذا الحقل علي ملاحظات من المبرمج وتعليقات علي الأمر الحالي وهو عادة ما يقوم بتوضيح وظيفة الأمر وأي معلومات إضافية قد تكون مفيدة لأي شخص قد يقرأ البرنامج وتساعده في فهمه. يتم بدء هذا الحقل بالفاصلة المنقوطة ";" وأي عبارة تقع بعد هذه الفاصلة المنقوطة يتم تجاهلها علي أنها ملاحظات.

رغم أن هذا الحقل اختياري ولكن لأن لغة التجميع تحتاج التعليمات فيها لبعض الشرح فإنه من الأفضل أن يتم وضع تعليقات علي أي أمر غير واضح أو يحتاج

لتفسير وعادة ما يتم وضع تعليق علي كل سطر من أسطر البرنامج ويتم اكتساب الخبرة بمرور الزمن عن كيفية وضع التعليق المناسب. فمثلاً التعليق التالي غير مناسب :

```
; move 0 to CX MOV  CX , 0
```

وكان من الأفضل أن يتم كتابة التعليق التالي :

```
;    CX    counts MOV  CX , 0
terms, initialized to 0
```

كما يتم أحياناً استخدام سطر كامل علي أنه تعليق وذلك في حالة شرح فقرة محددة كما في المثال التالي:

```
;
; Initialize Registers
MOV   CX,0
MOV      BX, 0
```

البيانات المستخدمة في البرنامج Program Data

يقوم البرنامج بالتعامل مع البيانات في صورة أرقام ثنائية وفي برامج لغة التجميع يتم التعامل مع الأرقام في الصورة الثنائية أو السداسية عشر أو العشرية أو حتى في صورة حروف.

الأعداد Numbers

- يتم كتابة الأرقام الثنائية في صورة 0 و1 وتنتهي الحرف B أو b للدلالة علي أن الرقم ثنائي Binary
- مثل 01010111B أو 11100011b
- الأرقام العشرية يتم كتابتها في الصورة المعتادة وبدون حرف في النهاية، كما يمكن أن تنتهي بالحرف D أو الحرف d دلالة علي أنها عشرية Decimal مثل 1234 و 1345d و –234D.
- الأرقام السداسية عشر يجب أن تبدأ برقم وتنتهي بالحرف H أو الحرف h للدلالة علي أنها سداسية عشر Hexadecimal مثل 0abh أو 56H . (السبب في استعمال 0 في المثال الأول لتوضيح أن المطلوب هو الرقم السداسي عشر ab وليس المتغير المسمي ab).

الجدول التالي يوضح بعض الأمثلة

الرقم	ملحوظات
10011	عشري
10011 b	ثنائي
6455	عشري
-456h	سداسي عشر

FFFF h	خطأ (لا يبدأ برقم)
1,234	خطأ (يحتوي على حرف غير رقمي)
0ab	خطأ (لم ينتهي بالحرف h أو H)

الحروف Characters

يتم وضع الحروف والجمل داخل علامات التنصيص مثلاً 'A' أو 'SUDAN' ويتم داخلياً تحويل الحروف إلي الأرقام المناظرة في كود الـ ASCII بواسطة الـ Assembler وبالتالي تخزينها في الذاكرة وعلي ذلك لا يوجد فرق بين الحرف 'A' والرقم 41h (وهو الرقم المناظر للحرف A في الجدول) وذلك داخل البرنامج أو من ناحية التخزين في الذاكرة.

المتغيـــرات VARIABLES

تلعب المتغيرات في لغــة التجميع نفس الدور الذي تلعبه في البرامج باللغات ذات المستوى العالي High Level Programming Languages مثل لغة الباسكال والسى. وعلى ذلك يجب تحديد أسماء المتغيرات

المستخدمة في البرنامج ونوع كل متغير حيث سيتم حجز مكان في الذاكرة لكل متغير وبطول يتناسب مع نوع المتغير وذلك بمجرد تعريف المتغير. ويتم استخدام الجدول التالي لتعريف المتغيرات في لغة التجميع حيث يشير كل إيعاذ لنوع المتغير المطلوب تعريفه.

الايعـــــــاز	المعـــــــــــــــــــــنـــى
DB (Define Byte)	لتعريف متغير حرفي يشغل خانة واحدة في الذاكرة
DW (Define Word)	لتعريف متغير كلمة يشغل خانتين متتاليتين في الذاكرة
DD (Define Double Word)	لتعريف متغير يشغل أربعة خانات متتالية في الذاكرة
DQ (Define Quad Word)	لتعريف متغير يشغل ثمان خانات متتالية في الذاكرة
DT (Define Ten Bytes)	لتعريف متغير يشغل عشر خانات متتالية في الذاكرة

في هذا الجزء سنقوم بالتعامل مع المتغيرات من النوع DB و DW .

المتغيرات الحرفيـــة Byte Variables :

يتم تعريف المتغيرات الحرفية بالصورة التالية:

Name DB Initial_Value

مثلاً

Alpha DB 4

يقوم هذا الإيعاز بتعريف متغير يشغل خانه واحدة في الذاكرة واسمه Alpha ويتم وضع قيمه ابتدائية مقدارها 4 في هذا المتغير .

يتم استعمال علامة الاستفهام (؟) في حالة عدم وجود قيمه ابتدائية للمتغير .

مثال : Byte DB ?

القيم التي يمكن تخزينها في هذا المتغير تتراوح بين 0 و 255 في حالة الأرقام التي يتم تخزينها بدون إشارة Unsigned Numbers وبين -128 و +127 في حالة الأرقام التي يتم تخزينها بإشارة Signed Numbers .

متغيرات الجمل Word Variables

يتم تعريف المتغير علي أنه من النوع Word ويتم تخزينه في خانتين من الذاكرة Two Bytes وذلك باستخدام الصيغة

name DW initial_value

مثلاً التعريف التالي

-2 DW WRD

يتم فيه تعريف متغير باسم WRD ووضع قيمة ابتدائية (الرقم –2) فيه

كما في حالة المتغيرات الحرفية يتم وضع العلامة ؟ في حالة عدم وجود قيمة ابتدائية للمتغير.

يمكن للمتغير من النوع word تخزين أرقام تتراوح بين 0 و 65535 (2^{16} –1) في حالة الأرقام بدون إشارة (الموجبة فقط)Unsigned Numbers ويمكن تخزين الأرقام من –32768 (-2^{15}) وحتى 32767 (2^{15} - 1) في حالة الأرقام بإشارة (الموجبة والسالبة) Signed Numbers.

المصفوفات Arrays

في لغة التجميع نتعامل مع المصفوفات علي أنها مجموعة من الحروف أو الكلمات المتراصة في الذاكرة في عناوين متتالية. فمثلاً لتعريف مصفوفة تحتوي علي ثلاثة أرقام من النوع الحرفي 3Bytes بقيم ابتدائية 10h و 20h و 30h علي الترتيب يتم استخدام التعريف التالي:

10h, 20h, 30h DB B_ARRAY

الاسم B_ARRAY يشير إلي العنصر الأول في المصفوف(العدد 10h) والاسم B_ARRAY + 1 يشير إلي العنصر الثاني والاسم B_ARRAY + 2 يشير إلي العنصر الثالث. فمثلاً إذا تم تخصيص عنوان الإزاحة 0200h للمتغير B_ARRAY يكون شكل الذاكرة كما يلي:

الاسم (الرمز (Symbol	العنوان	المحتوي
B_ARRAY	0200h	10h
B_ARRAY + 1	0201h	20h
B_ARRAY + 2	0202h	30h

وبنفس الطريقة يتم تعريف مصفوف مكون من كلمات فمثلاً التعريف

W_ARRAY DW 1000h, 2000h, 3000h

يقوم بتعريف مصفوف يحتوي علي ثلاثة عناصر بقيم ابتدائية 1000h و 2000h و 3000h علي الترتيب. يتم تخزين القيمة الأولي (1000h)في العنوان W_ARRAY والقيمة الثانية في العنوان W_ARRAY +2 والقيمة الثالثة في العنوان W_ARRAY + 4 وهكذا. فمثلاً لو تم تخزين المصفوف في

الذاكرة بدءاً من العنوان 300h يكون شكل الذاكرة كما يلي :

الاسم (الرمز (Symbol	العنوان	المحتوي
W_ARRAY	0300h	1000h
W_ARRAY + 2	0302h	2000h
W_ARRAY + 4	0304h	3000h

لاحظ أن للمتغيرات من هذا النوع يتم تخزينها في الذاكرة في خانتين حيث يتم تخزين الخانة ذات الوزن الأقل Low Byte في الخانة الأولي والخانة ذات الوزن الأكبر High Byte في العنوان التالي مباشرة. فمثلاً إذا كان لدينا التعريف : Word1 DW 1234h

يتم تخزين الرقم 34h (الذي يمثل الخانة ذات الوزن الأقل) في العنوان word1 والرقم 12h (الذي يمثل الخانة ذات الوزن الأكبر) في العنوان word1 + 1.

الرسائل والنصوص Character Strings

يتم تخزين النصوص علي أنها سلسلة من الحروف ويتم وضع القيمة الابتدائية في صورة حروف أو القيم المناظرة للحروف في جدول الحروف ASCII Table فمثلاً التعريفان التاليان التاليين يؤديان إلي نفس النتيجة وهي تعريف متغير اسمه Letters ووضع القيمة الابتدائية "ABC' فيه

```
1 - Letters     db   'ABC'
2 – Letters     db   41h, 42h,43h
```

ويمكن دمج القيمة الابتدائية لتحوي الحروف والقيم المناظرة لها كما في المثال التالي

```
msgdb    0dh,0ah,'Sudan$'
```

ويتم هنا بالطبع التفرقة بين الحروف الكبيرة Capital Letters والحروف الصغيرة Small Letters.

الثوابت

يتم عادة استخدام الثوابت لجعل البرنامج أسهل من حيث القراءة والفهم وذلك بتعريف الثوابت المختلفة المستخدمة في البرنامج. يتم استخدام الإيعاز EQU (EQUate) لتعريف الثوابت علي النحو التالي :

```
name       EQU    Constant
```

حيث name هو اسم الثابت. مثلاً لتعريف ثابت يسمي LF بقيمة ابتدائية 0Ah نكتب

```
LF   EQU      0Ah
```

وبالتالي يمكن استخدام الثابت LF بدلاً عن الرقم 0Ah كالآتي MOV AL , LF بدلاً عن استخدام الآتي MOV AL,0Ah . حيث يقوم الـ Assembler بتحويل الثابت LF داخل البرنامج إلي الرقم 0Ah.

كذلك يمكننا استخدام المثال التالي

```
Prompt      EQU      'Type your Name'
Msg         DB  prompt
```

لاحظ أن EQU عبارة عن إيعاز وليس تعليمه أو أمر وبالتالي لا ينتج عنه تعريف متغير ووضعه في الذاكرة.

بعض الأوامر الأساسية

في هذا الجزء سنتعرف علي بعض الأوامر الأساسية وكيفية استخدامها والقيود المتلفة علي استخدامها وسنفترض أن لدينا متغيرات حرفية باسم Byte1 و Byte2 ومتغيرات كلمة باسم Word1 و Word2

1 – الأمر MOV

يستخدم الأمر MOV في نقل البيانات من مكان لآخر وهذه الأماكن هي المسجلات العامة أو المسجلات الخاصة أو المتغيرات في الذاكرة أو حتى في نقل (وضع) قيمة ثابتة في مكان محدد من الذاكرة أو على مسجل.

والصورة العامة للأمر هي

MOV Destination , Source

حيث يتم نقل محتويات المصدر Source إلي المستودع Destination ولا تتأثر قيمة المصدر بعد تنفيذ الأمر

مثلاً

MOV AX , Word1

حيث يتم نسخ محتويات (قيمة) المتغير Word1 إلي المسجل AX. وبالطبع يتم فقد القيمة الأولية للمسجل AX بعد تنفيذ الأمر. كذلك الأمر

MOV AL, ‘A’

يقوم بوضع الرقم 041h (وهو الرقم المناظر للحرف A في جدول الـ ASCII) في المسجل AL.

الجدول التالي يوضح قيود استخدام الأمر MOV

المصدر	المستودع			
	مسجل عام	مسجل مقطع	متغير (موقع في الذاكرة)	ثابت
مسجل عام	مسموح	مسموح	مسموح	غير مسموح
مسجل مقطع	مسموح	غير مسموح	مسموح	غير مسموح
متغير (موقع في الذاكرة)	مسموح	مسموح	غير مسموح	غير مسموح
ثابت	مسموح	غير مسموح	مسموح	غير مسموح

2- الأمر XCHG (Exchange)

يستخدم الأمر XCHG لاستبدال قيمة مسجلين أو لاستبدال قيمة مسجل مع موقع محدد في الذاكرة (متغير). والصيغة العامة للأمر هي:

XCHG Destination, Source

مثال:

XCHG AH, BL

حيث يتم تبادل قيم المسجلين AH, BL (تصبح قيمة AH تساوى قيمةBL وBL تساوى AH).

مثال:

الأمر التالي يقوم باستبدال قيمة المسجل AX مع المتغير WORD1

XCHG AX, WORD1

الجدول التالي يوضح قيود استخدام الأمر XCHG

	المستودع	
المصدر	مسجل عام	موقع في الذاكرة
مسجل عام	مسموح	مسموح
موقع في الذاكرة	مسموح	غير مسموح

3 – العمليات الحسابية ADD, SUB, INC, DEC, NEG:

يتم استخدام الأمرين ADD و SUB لجمع أو طرح محتويات مسجلين أو مسجل وموقع في الذاكرة أو موقع

في الذاكرة مع مسجل أو مسجل مع موقع في الذاكرة

والصيغة العامة للأمرين هي:-

Destination, Source ADD

SUB Destination, Source

مثلاً الأمر

ADD WORD1, AX

يقوم بجمع محتويات المسجل AX إلي قيمة المتغير WORD1 ويتم تخزين النتيجة في المتغير WORD1 (لا يتم تغيير قيمة محتويات المسجل AX بعد تنفيذ الأمر)

كذلك الأمر

SUB AX, DX

حيث يتم طرح محتويات المسجل DX من المسجل AX ويتم تخزين النتيجة في المسجل AX (لاحظ أن محتويات المسجل DX لا تتغير بعد تنفيذ الأمر)

الجدول التالي يبين قيود استعمال الأمرين ADD و SUB

المصدر	المستودع	
	مسجل عام	موقع في

		الذاكرة
مسجل عام	مسموح	مسموح
موقع في الذاكرة	مسموح	غير مسموح
ثابت	مسموح	مسموح

لاحظ أنه غير مسموح بالجمع أو الطرح المباشر بين مواقع في الذاكرة في أمر واحد وبالتالي فإن الأمر ADD BYTE1, BYTE2 غير مسموح به ولكن يمكن إعادة كتابته على الصورة:

MOV AL, BYTE2 ; حيث يتم قيمة المتغير الي مسجل قبل عملية الجمع

ADD BYTE1, AL

الأمر ADD BL,5 يقوم بجمع الرقم 5 إلي محتويات المسجل BL وتخزين النتيجة في المسجل BL.

كملاحظة عامه نجد انه يجب أن يكون المتغيرين لهما نفس الطول بمعني أن الأمر التالي غير مقبول

MOV AX ,BYTE1

وذلك لأن طول المتغير BYTE هو خانه واحدة أما المسجل AX فان طوله هو خانتين 2-BYTE. (أي أن المتغيرات (المعاملات) يجب أن تكون من نفس النوع)

بينما نجد الـ ASEMBLER يستقبل الأمر MOV AH, ‘A’ (مادام AH بايت فإن المصدر يجب أن يكون كذلك بايت)

حيث يتم وضع الرقم 41h في المسجل AH ويقوم أيضا بتقبل الأمر

MOV AX ,’A’ (مادام AX كلمة فإن المصدر يجب أن يكون كذلك كلمة)

حيث سيتم وضع الرقم 0041h في المسجل AX .

الأوامر INC (Increment) , DEC (Decrement) , NEG

أما الأمرين INC ,DEC يتم فيها زيادة أو نقصان قيمه مسجل أو موقع في الذاكرة بمقدار 1 والصيغة العامة لها هي:

INC Destination ; Destination = Destination +1

DEC Destination ; Destination = Destination - 1

فمـثلا الأمر INC WORD1 يقـوم بجمع 1 إلى محتويات المتغيـر WORD1

بيـنما الأمر DEC WORD2 يـقوم بإنقاص الرقم 1 مـن محتويات المتغيـر WORD2 .

أخيراً نتحدث عن الأمر NEG(Negate) والذي يستعمل لتحويـل إشارة الرقم الموجب إلي رقم سالب والـرقم السالب يـتم تحويـله إلي رقم موجب وذلك بتحويـلـه إلى المكمل لاثنين 2'S Complement والصيغة العامة للأمر هي :

NEG Destination

حيـث يتـم التعـامل مـع أحد المسجلات أو مـوقع في الذاكرة

مثـال :

```
NEG   BX          ;  BX =  -BX
NEG   BYTE        ;  BYTE = - BYTE .
```

تحويل العبارات إلي صورة برامج التجميع:-

لكي يتم التعامل مع الأوامر السابقة سنقوم في هذا الجزء بتحويل بعض العمليات من لغات البرمجة العليا

إلي High Level Programming Languages

تعليمات بلغة التجميع .

إذا افترضنا أن المتغيرين A و B عبارة عن

متغيرين من النوع WORD.

لتحويل العبارة B=A

لأنه لا يمكن نقل محتويات لمتغير في الذاكرة إلي

متغير آخر في الذاكرة مباشرةً يلزم تحويل العبارة إلي

نقل قيمة المتغير إلي مسجل ثم نقل قيمة المسجل إلي

الرقم المطلوب

انقل محتويات A الي المسجل AX قبل نقلها الى B

MOV AX , A

MOV B , AX

أما الأمر A=5-A يتم تحويلة إلى الأوامر

ضع 5 في AX MOV AX , 5

AX تحتوي علي 5-A SUB AX , A

ضعها في A MOV A , AX

أو إلى الأوامر

NEG A

ADD A,5

وأخيراً الأمر A=B-2*A يتم تحويلة إلى الأوامر

MOV AX,B

SUB AX,A
SUB AX, A
MOV A,AX

الشكل العام للبرنامج:-

في الفصل السابق قمنا بتوضيح عملية تقسيم الذاكرة إلى مقاطع مختلفة بحيث يحتوى المقطع الأول علي البرنامج نفسه ويسمى مقطع البرنامج CODE SEGMENT ومقطع آخر يحتوى علي البيانات المستخدمة في البرنامج ويسمى مقطع البيانات DATA SEGMENT ومقطع ثالث يحتوي علي المكدس ويسمى مقطع المكدس STACK SEGMENT

في هذا الجزء سيتم توضيح كيفية توليد هذه المقاطع بواسطة الـ ASSEMBLER مع توضيح كيفية كتابة وتعريف كل مقطع داخل البرنامج.

نماذج الذاكرة MEMORY MODELS:

كما ذكرنا فيما مضى انه قد يكون البرنامج المطلوب كتابته صغير بحيث يمكن أن يسع مقطع واحد فقط لكل من البرنامج والبيانات والمكدس وقد تحتاج إلي

استخدام مقطع منفصل لكل على حده. يتم استعمال الكلمة MODEL. وذلك بكتابة السطر التالي :

.MODEL
MEMORY_MODEL

ويتم كتابة هذا السطر قبل تعريف أي نقطة ويوجد لدينا اكثر من نموذج للذاكرة سوف يتم توضيحها في الجدول التالي ولكن عموماً إذا لم يكن حجم البيانات كبيراً يتم غالباً استخدام النموذج SMALL وهذا هو الحال في اغلب البرامج التي سنتطرق لها. ويتم كتابة السطر على الصورة التالية: .MODEL SMALL

الجدول التالي يوضح أسماء موديلات الذاكرة المختلفة وتوضيح خصائص كل منها

الموديل MODEL	الوصف
SMALL	الكود في مقطع واحد والبيانات في مقطع واحد
MEDIUM	الكود في اكثر من مقطع والبيانات في مقطع واحد
COMPACT	الكود في مقطع واحد والبيانات في اكثر من مقطع

LARGE	الكود في اكثر من مقطع والبيانات في اكثر من مقطع ولكن غير مسموح بتعريف مصفوفة اكبر من 64k BYTE
HUGE	الكود في اكثر من مقطع والبيانات في اكثر من مقطع ولكن يمكن أن يكون هناك مصفوفة بطول اكبر من 64k BYTE

مقطع البيانات DATA SEGMENT:

يحتوى مقطع البيانات على تعريف كل المتغيرات وبالنسبة للثوابت يمكن تعريفها في في مقطع البيانات أو في أي مكان آخر نسبة لأنها لا تشغل مكان في الذاكرة.

لتعريف مقطع البيانات يتم استخدام التعريف DATA. وبعد ذلك يتم تعريف المتغيرات والثوابت مباشرة والمثال التالي يوضح ذلك

```
.DATA
WORD1    DW   2
WORD2    DW   5
MSG      DB  'THIS IS A MESSAGE'
```

MASK EQU 10011001B

مقطع المكدس Stack Segment :

الغرض من مقطع المكدس هو حيز جزء من الذاكرة ليتم استخدامه في عملية تكديس البيانات أثناء تنفيذ البرنامج. ويجب أن يكون هذا الحجم كافي لتخزين كل المكدس في أقصي حالاته (لتخزين كل القيم المطلوب تكديسها أثناء عمل البرنامج).

ويتم تعريف مقطع المكدس باستخدام التعريف:

.Stack Size

حيث size يمثل عدداً إختيارياً هو حجم المكدس بالوحدات bytes. والمثال التالي يقوم بتعريف المكدس بحجم 100h

.Stack 100h

إذا لم يتم تعريف الحجم يتم افتراض الحجم 1KB بواسطة الـ Assembler .

مقطع البرنامج Code Segment :

يحتوى هذا المقطع على الأوامر والتعليمات المستخدمة داخل البرنامج ويتم تعريفه على النحو التالي:

.Code Name

حيث Name هو اسم المقطع .ولا داعي لإعطاء اسم للمقطع في حالة النموذج Small (لان لدينا مقطع واحد

فقط) حيث سيقوم برنامج الـ Assembly بإعطاء رسالة خطأ في هذه الحالة.

داخل مقطع البرنامج يتم وضع الأوامر في صورة برامج صغيـــرة (إجراءات) Procedure وأبسط تعريف لهذه الإجراءات على النحو التالي

```
Name      Proc
;الأوامر والتعليمات داخل الإجراء
Name    ENDP
```

حيث Name هو اسم الإجراء، أما Proc و Endp فهما إيعازات Pseudo_Ops

الجزء التالي يوضح مقطع برنامج كامل

```
.CODE
MAIN PROC
;الأوامر والتعليمات داخل الإجراء
MAIN   ENDP
;بقية الإجراءات يتـــم كتابتهـــــا هـــــنا
```

والآن بعد أن رأينا كل مقاطع البرنامج فان الشكل العام للبرنامج في حالة النموذج small. يكون على النحو التالي :

```
.MODEL SMALL
.STACK 100H
```

.DATA

هنا يكون تعريف المتغيرات والثوابت ;

.CODE

MAIN PROC

التعليمات والأوامر داخل الإجراء ;

MAIN ENDP

بقية الإجراءات تكتب هنا;

END MAIN

آخر سطر في البرنامج يحوى كلمة نهاية البرنامج ENDمتبوعة باسم الإجراء الرئيسي في البرنامج .

تعليمات الإدخال والإخراج INPUT &OUTPUT INSTRUCTIONS

يتعامل المعالج الدقيق مع الأجهزة الخارجية باستخدام موانئ الإدخال والإخراج وذلك باستخدام الأوامر IN للقراءة وفى ميناء إدخال والأوامر OUT للكتابة في ميناء إخراج . ويتم استخدام هذه الأوامر في بعض الأحيان بالذات إذا كان المطلوب هو سرعة التعامل مع الجهاز الخارجي وعادة لا يتم استخدام هذه الأوامر في البرامج التطبيقية لسببين الأول أن عناوين الموانئ قد تختلف من جهاز لآخر مما يتطلب تعديل البرنامج في كل

مرة ، والثاني انه من الأسهل التعامل مع الأجهزة الخارجية بواسطة الشركات المصنعة للأجهزة بواسطة روتينات خدمة SERVICE ROUTINES يتم توفيرها بواسطة الشركات المصنعة للأجهزة .

يوجد نوعان في روتينات الخدمة المستخدمة في التعامل مع الموانئ يسمى الأول BIOS (BASIC INPUT /OUTPUT SYSTEM) والثاني باستخدام الـ DOS. روتينات الـ BIOS يتم تخزينها في ذاكرة القراءة فقط (الـ ROM) ويتعامل مباشرة مع موانئ الإدخال والإخراج بينما خدمات الـ DOS تقوم بتنفيذ عمليات اكثر تعقيداً مثلاً طباعة سلسلة حروف وهي تقوم عادة باستخدام ال BIOS في تنفيذ عمليات إدخال/إخراج مباشرة.

يتم نداء الـ BIOS أو الـ DOS لتنفيذ عملية محددة باستخدام نداء مقاطعة (INT (INTERRUPT والنداء على هذه الصورة

INT INTERRUPT_NUMBER

حيث يتم تحديد رقم نداء المقاطعة وهو رقم محدد مثلاً INT 16h يقوم بطلب خدمة في الـ BIOS وهى خاصة بقراءة قيمة في لوحة المفاتيح و INT 21h خاص

بنداء خدمة من الـ DOS سيتم التعرف على مزيد من الخدمات لاحقاً بإذن الله

نداء المقاطع رقم21H (INT 21H)

يتم استخدام هذا النداء لتنفيذ مجموعة كبيرة من الخدمات التي يقدمها نظام التشغيل DOS حيث يتم وضع رقم الخدمة المطلوبة في المسجل AH وقد يتطلب الأمر وضع بعض القيم في مسجلات أخرى وذلك حسب نوع الخدمة المطلوبة وبعد ذلك يتم نداء طلب المقاطعة 21H .وقد يتطلب الأمر استقبال قيم محددة في نداء المقاطعة حيث يتم وضعها في المسجلات .يتم وضع الخدمات المختلفة في جدول كبير يوضح وظيفة كل خدمة والمدخلات إليها والمخرجات منها.

الجدول التالي يوضح ثلاثة فقط من الخدمات التي يخدمها النظام

رقم الخدمة	الوصف (الروتين)

1	قراءة قيمة واحدة من لوحة المفاتيح
2	كتابة حرف واحد في الشاشة
9	كتابة مجموعة من الحروف في الشاشة

في الجزء التالي ستناول بعض هذه الخدمات

الخدمة رقم 1 : قراءة حرف من لوحة المفاتيح

المدخلات : وضع الرقم 1 في المسجل AH

المخرجات : المسجل AL يحتوي علي كود ال ASCII للحرف الذي تم الضغط عليه في لوحة المفاتيح أو 0 في حالة الضغط على مفتاح غير حرفي NON CHARACHTER KEY

(مثلا المفاتيح F1-F10).

لتنفيذ هذه الخدمة يتم كتابة الآتي:-

```
MOV    AH,01
INT 21H
```

تقوم هذه الخدمة بانتظار المستخدم إلى حين الضغط على لوحة المفاتيح . عند الضغط على أي

مفتاح يتم الحصول على كود الـ ASCII للمفتاح من المسجل AL كما يتم عرض الحرف الذي تم الضغط عليه في لوحة المفاتيح علي الشاشة. ولا تقوم هذه الخدمة بإرسال رسالة إلي المستخدم فهي فقط تنتظر حتى يتم الضغط على مفتاح. إذا تم ضغط بعض المفاتيح الخاصة مثل F1-F10 فسوف يحتوي المسجل AL علي القيمة صفر . التعليمات التي تلي INT 21h تستطيع فحص المسجل AL و تتخذ الفعل المناسب.

2- الخدمة رقم 2 : عرض حرف على الشاشة أو تنفيذ وظيفة تحكم.

المدخلات : وضع الرقم 02 في المسجل AH .

وضع شفرة الـ ASCII كود للحرف المطلوب عرضه في المسجل DL .

المخرجات : الكود الـ ASCII للحرف الذي تم عرضه يتم وضعه في المسجل AL.

مثال: الأوامر التالية تعرض علامة استفهام علي الشاشة

```
MOV     AH , 02H
MOV     DL , '?'
INT 21H
```

بعد طباعة الحرف على الشاشة يتحرك المؤشر

إلي الموضع التالي (إذا كان الوضع

الحالي هو نهاية السطر يتحرك المؤشر إلي بداية

السطر الجديد).

يتم استخدام هذه الخدمة لطباعة حرف التحكم

Control Character أيضاً والجدول التالي يوضح

بعض حروف التحكم)

الكود ASCII	الرمز	الوظيفة
7	BEL (Beep)	إصدار صوت
8	BS (Back space)	مسافة للخلف Back (Space)
9	HT (Tab)	تحرك بمقدار Tab
A	LF (Line Feed)	سطر جديد
D	CR (Carriage return)	بداية السطر الحالي

بعد التنفيذ يحصل المسجل AL علي شفرة ASCII

لحرف التحكم

البرنامج الأول:

برنامجنا الأول سيقوم بقراءة حرف من لوحة المفاتيح ثم طباعة الحرف الذي تم إدخاله في بداية السطر التالي ثم إنهاء البرنامج.

يتكون البرنامج من الأجزاء التالية:

1- إظهار علامة الاستفهام "؟" على الشاشة

```
MOV     AH,2
MOV     DL,'?'
INT 21h
```

2- قراءة حرف من لوحة المفاتيح

```
MOV     AH,1
INT 21h
```

3- حفظ الحرف الذي تم إدخاله في مسجل آخر BL مثلاً و ذلك لأننا سنستخدم المسجل DL في تحريك المؤشر إلي بداية السطر الجديد وسيؤدي ذلك لتغيير محتويات المسجل AL (لاحظ أن الخدمة 2 تقوم باستقبال الحرف المطلوب طباعته في المسجل DL وتقوم بإعادة الحرف المطبوع في المسجل AL مما يجعلنا نفقد القيمة المسجلة فيه) وبالتالي يجب تخزين محتوياته في مسجل آخر مثل BL

```
MOV     BL , AL
```

4- لتحريك المسجل إلي بداية السطر الجديد يجب طباعة حرف التحكم

Carriage Return و Line Feed ويتم ذلك كالآتي

MOV AH,2
MOV DL,0dh ;
Carriage Return
INT 21h
MOV DL,0ah ; Line
Feed
INT 21h

5- طباعة الحرف الذي تم إدخاله (لاحظ انه تم تخزينه في المسجل BL في الخطوة (3)

MOV DL , BL
INT 21h

6- إنهاء البرنامج و العودة الى نظام التشغيل ويتم ذلك بوضع الرقم 4Ch في المسجل AH

واستدعاء نداء المقاطعة رقم 21h.

MOV AH,4CH
INT 21h

و على ذلك يصبح البرنامج على الصورة التالية :

```
TITLE   FIRST : ECHO PROGRAM
.MODEL   SMALL
.STACK   100H
.CODE
MAIN   PROC
```

اظهار علامة التعجب ;

```
MOV   AH,2            ; طباعة حرف
MOV   DL,'?'          ; الحرف المطلوب طباعته
INT    21H
;قراءة حرف من لوحة المفاتيح
MOV   AH,01           ; قراءة حرف
INT    21H
MOV   BL,AL           ; تخزين الحرف
; الذهاب إلى سطر جديد
MOV   AH,02
MOV   DL,0DH   ;   carriage return
INT   21H
MOV   DL,0AH   ;   line  feed
INT   21H
; طباعة الحرف الذي تم إدخاله
MOV   DL,BL           ; ; إحضار الحرف من المسجل
INT   21H
; DOS العودة إلى نظام التشغيل
MOV   AH,4CH
INT   21H
```

MAIN ENDP
END MAIN

لاحظ أنه عندما يتوقف البرنامج فإنه يحول التحكم للـ DOS بتنفيذ INT 21h الوظيفة 4Ch

ولأنه لم يتم استخدام المتغيرات فقد حذف قطاع البيانات في هذا البرنامج

إنشاء وتشغيل البرنامج :-

في هذا الجزء سنوضح طريقة إنشاء و تجهيز البرنامج للتشغيل حيث يتضمن ذلك الخطوات التالية:-

1- استخدام أي برنامج Text Editor لكتابة البرنامج الموضح في المثال السابق. (ملف برنامج المصدر)

2- استخدام الـ ASSEMBLER لتوليد الملف المسمى OBJECT FILE .

3- استخدام برنامج الربط LINKER لربط ملفات الـ OBJECTلتوليد ملف التشغيل EXECUTABLE FILE .

4- تشغيل البرنامج.

فيما يلي توضيح بالتفصيل كل خطوة من الخطوات السابقة:-

1 - إنشاء ملف البرامج SOURCE FILE :-

يتم استخدام أي محرر نصوص Editor لكتابة البرنامج ويمكن استخدام أي محرر ينتج ملف نصي عادى Text Editor مثل EDIT يتم عادة تخزين الملف بامتداد ASM (Extention) مثلا المثال السابق نحفظ الملف بالاسم FIRST.ASM .

2- تجميع البرنامج ASSEMBLE THE PROGRAM:-

ويتم هذا عن طريق معالجة البرنامج بواسطة أحد الـ Assembler مثل MASM(Microsoft Macro Assembler) أو TASM(Turbo Assembler) و التي تقوم بتحويل الملف الأصلي الذي يحتوى على البرنامج المكتوبة بلغة التجميع إلى ملف اقرب إلى لغة الآلة يسمى(OBJECT FILE) . وأثناء هذه العملية يتم التعامل مع الملف والتأكد من عدم وجود أي خطأ في كتابة البرنامج حيث يتم الرجوع إلي الخطوة (1) وتحديد الأخطاء و تصحيحها حتى نحصل على رسالة بعدم وجود أخطاء في البرنامج.

واستخدام البرنامج TASM أوMASM يتم على النحو التالي :

TASM FILENAME;

أو   MASM FILENAME;

في هذا الجزء سنستخدم برنامج TASM والجزء التالي

يوضح هذه العملية:-

>TASM FIRST;
TURBO ASSEMBLER VERSION 3.1
COPYRGHT(C)1988,1992BRLAND
INTERNATIONAL
ASSEMBLING FILE : FIRST.SAM
ERROR MESSAGE : NONE
WARNING MESSAGE :NONE
PASSES: 1

السطر الأول يوضح نوع الـ ASSEMBLER والسطر الثاني يوضح اسم الملف يليه سطرين بالأخطاء التي توجد في البرنامج.

لاحظ أنه إذا كان هناك أي خطأ في البرنامج الأصلي يتم إظهار رسالة تحوي رقم السطر ونبذة سريعة عن الخطأ حيث يجب فتح الملف الأصلي first.asm وتصحيح الخطأ ثم العودة مرة أخرى وإعادة هذه الخطوة حتى نحصل على الملف first.obj .

3-ربط البرنامج Linking the program

الملف الذي تم إنشاؤه في الخطوة السابقة هو ملف بلغة الآلة Machine Language ولكنه غير قابل للتنفيذ لأنه لا يحتوي على الشكل المناسب للبرامج القابلة للتنفيذ وذلك للأسباب التالية:

أ- عدم تعريف مكان تحميل الملف في الذاكرة وبالتالي فإن عمليه العنونة داخل البرنامج لا يمكن تنفيذها.

ب- بعض الأسماء والعناوين داخل البرنامج تكون غير معرفة بالذات في حالة ربط أكثر من برنامج حيث يتم من أحد البرامج نداء برامج فرعيه أخرى مكتوب في ملف آخر.

برنامج الربط Link Program يقوم بإجراء عملية الربط بين الـ Object Files المختلفة وتحديد العناوين داخل البرنامج ويقوم بعد ذلك بإنتاج ملف قابل للتنفيذEXE . (Executable File) على النحو التالي :

> TLINK First;
Turbo Link Version 2.0 Copyright (c) 1987
Borland International .

4 – تنفيذ البرنامج Run The Program

لتشغيل البرنامج يتم فقط كتابة اسمه من محث الـDOS

C:\ASM > first
?t
t
C:\ASM >

يقوم البرنامج بطباعة الحرف "؟" والانتظار إلي حين الضغط علي مفتاح من لوحة المفاتيح. يقوم البرنامج بالذهاب إلي بداية السطر الجديد وطباعة الحرف الذي تم الضغط عليه ثم الانتهاء والعودة إلي نظام التشغيل.

إظهار رسالة علي الشاشة Display String

في البرنامج السابق تم استخدام الوظيفة رقم 1 من نداء المقاطعة رقم 21h وهي تستخدم لاستقبال حرف من لوحة المفاتيح وكذلك الوظيفة رقم 2 وهي لطباعة حرف علي الشاشة.

في هذا المثال ولإظهار رسالة كاملة علي الشاشة يتم استخدام الخدمة رقم 9

خدمة رقم 9 : إظهار رسالة علي الشاشة

المدخلات : عنوان الإزاحة Offset لبداية الرسالة يتم وضعه في المسجل DX

(يجب أن تنتهي الرسالة بالحرف "$")

الحرف "$" في نهاية الرسالة لا تتم طباعته علي الشاشة. وإذا احتوت الرسالة علي أي حرف تحكم Control Character فإنه يتم تنفيذه أثناء الطباعة.

لتوضيح هذه العملية سنقوم بكتابة برنامج يقوم بإظهار الرسالة 'Hello!' في الشاشة. يتم تعريف هذه الرسالة في مقطع البيانات بالطرقة التالية

```
msg db   'HELLO!$'
```

الأمر LEA

تحتاج الخدمة رقم 9 في نداء المقاطعة INT 21h إلي تجهيز عنوان إزاحة الرسالة في المسجل DX ولعمل ذلك يتم تنفيذ الأمر LEA (Load Effective Address)

```
LEA Destination , Source
```

حيث المستودع هو أحد المسجلات العامة والمصدر هو اسم المتغير الحرفي (موقع في الذاكرة). يقوم الأمر بوضع عنوان الإزاحة للمتغير المصدر في المسجل المستودع. فمثلاً الأمر

DX, MSGLEA

يقوم بوضع قيمة الإزاحة لعنوان المتغير msg في المسجل DX .

ولأن هذا البرنامج يحتوي علي مقطع بيانات فإننا نحتاج إلي تجهيز المسجل DS لكي يشير إلي مقطع البيانات.

بادئة مقطع البرنامج PSP (Program Segment Prefix)

عندما يتم تحميل البرنامج في الذاكرة يقوم نظام التشغيل بتخصيص 256 خانة للبرنامج وهي تسمي PSP. يحتوي الـ PSP علي معلومات عن البرنامج وعلي ذلك يستطيع البرنامج التعامل مع هذه المعلومات. يقوم نظام التشغيل DOS بوضع عنوان المقطع الخاص به في كل من المسجلين DS و ES قبل تنفيذ البرنامج ونتيجة لذلك فإن مسجل مقطع البيانات DS لا يحتوي علي عنوان مقطع البيانات الخاص بالبرنامج ولعلاج هذه المشكلة فإن أي برنامج يحتوي علي مقطع بيانات يجب أن يبدأ بتجهيز مسجل مقطع البيانات ليشير إلي مقطع البيانات الخاص بالبرنامج علي النحو التالي

```
AX,      MOV
@DATA
MOV      DS, AX
```

حيث DATA@ هو عنوان مقطع البيانات الخاص بالبرنامج والمعرف بـ DATA. حيث يقوم الـ ASSEMBLER بتحويل الاسم DATA@ إلي رقم يمثل عنوان المقطع ولأننا لا نستطيع تخزين النتيجة في المسجل DS مباشرة فقد استعنا بمسجل عام AX كمسجل وسيط يتم وضع القيمة فيه أولاً وبعد ذلك يتم نقلها إلي المسجل DS.

بعد ذلك يمكن طباعة الرسالة '!HELLO' وذلك عن طريق وضع عنوانها في المسجل DX واستخدام الخدمة رقم 9 في نداء المقاطعة رقم 21h. البرنامج التالي يوضح هذه العملية بالتفصيل

```
TITLE SECOND: DISPLAY STRING
.MODEL SMALL
.STACK 100H
.DATA
MSG DB 'HELLO!$'
.CODE
MAIN PROC
; initialize DS
MOV AX,@DATA
MOV DS,AX
;display message
```

```
LEA DX,MSG          ; احصل علي الرسالة
MOV AH,09H          ; وظيفة عرض السلسلة
INT 21H
;return to DOS
MOV AH,4CH
INT 21H             ; الخروج الي نظام التشغيل
MAIN    ENDP
END MAIN
```

برنامج تحويل حالة الحروف A Case Conversion Program:

في هذا المثال سنقوم بسؤال المستخدم ليقوم بإدخال حرف صغير lower-case letter يقوم البرنامج بإظهار رسالة تطبع الحرف الذي تم إدخاله بعد تحويله إلى صورة حرف كبير upper-case letter مثلاً

```
Enter A Lower Case Letter : a
In  Upper  Case  It  Is        : A
```

سيتم في هذا البرنامج استخدام الإيعاز EQU لتعريف كل من CR,LF

```
CR        EQU      0DH
LF        EQU      0AH
```

بينما يتم تعريف الرسائل على النحو التالي

```
MSG1    DB  'Enter A Lower Case Letter :$'
```

```
MSG2    DB  CR,LF,' In  Upper  Case  It
                                  Is        : '
Char      DB  ? ,'$'
```

عند تعريف المتغير char تم تعريفه بعد الرسالة MSG2 مباشرة وذلك لأن البرنامج سيقوم بإظهار الرسالة msg2 متبوعة مباشرة بالحرف char (وهو الحرف الذي تم إدخاله بعد تحويله إلى Upper -case ويتم ذلك بطريقة طرح الرقم 20h من الحرف الذي تم إدخاله)

تم تعريف حروف التحكم CR,LF قبل الرسالة msg2 بهدف جعل الرسالة تبدأ من بداية السطر الجديد.

ولأن الرسالة msg2 لا تنتهي بعلامة نهاية الرسالة '$' فإنه سيتم الاستمرار في الطباعة وطباعة الحرف char في الشاشة (لاحظ أن العلامة '$' توجد في نهاية المتغير char مباشرة).

يبدأ البرنامج بإظهار الرسالة msg1 ثم قراءة الحرف من لوحة المفاتيح

```
LEA DX ,msg1
MOV      AH ,9
INT 21h
MOV      AH ,1
INT 21h
```

upper-caseبعد ذلك يتم تحويل الحرف إلى حرف كبير
من الحرف (وذلك لأن الفرق بين 20hوذلك بطرح العدد
هو العدد ASCIIالحروف الكبيرة والصغيرة في جدول
بينما تبدأ 41h حيث تبدأ الحروف الكبيرة ابتداءً من 20h
(ويتم تخزين النتيجة في 61hالحروف الصغيرة ابتداءً من
charالمتغير

SUB AL,20h ; حوله الي حرف كبير
MOV char ,AL ; ثم خزِنهُ في المتغير

بعد ذلك يقوم البرنامج بإظهار الرسالة الثانية msg2 وتطبع متبوعة بالمتغير char كما ذكرنا سابقاً . وفيما يلي نص البرنامج :

```
TITLE THIRD: CASE CONVERSION PROGRAM
.MODEL SMALL
.STACK 100H
.DATA
CR  EQU     0DH
LF   EQU     0AH
MSG1    DB  'ENTER A LOWER CASE
LETTER: $'
MSG2    DB  CR,LF,'IN UPPER CASE IT IS :'
CHAR     DB  ?,'$'
.CODE
MAIN PROC
; initialize DS
```

```
MOV     AX,@DATA
MOV     DS,AX
;print user prompt
LEA DX,MSG1
MOV     AH,09H
INT 21H
;input character and convert to lower case
MOV     AH,01H
INT 21H
SUB     AL,20H
MOV     CHAR,AL
;display on the next line
LEA     DX,MSG2
MOV     AH,09H
INT 21H
;return to DOS
MOV     AH,4CH
INT 21H
MAIN    ENDP
END     MAIN
```

تمـــــــاريــــــــن :-

1- اذكر أي من الأسماء التالية صحيحاً وأيها خطأ في لغة التجميع الخاصة بـ IBM PC ولماذا ...؟

1- two_words
2- ?1
3- tow words

4- t=

2- أي من الأرقام التالية صحيح وأيها خطأ . وإذا كانت صحيحة اذكر نوع الرقم ثنائي عشري أو سداسي عشري.

1- 246   2- 246h   3- 1001   4- 1.101

5- 2EAH   6- FFEH   7-1011B

3- أعطِ تعريف كلٍ من المتغيرات التالية (إذا كان ممكناً)

أ-متغير كلمة word اسمه A وبه قيمة ابتدائية 52 .

ب-متغير كلمة word اسمه word1 ولا توجد به قيمة ابتدائية .

جـ-متغير حرف Byte اسمه B وبه قيمة ابتدائية 52 .

د-متغير حرف Byte اسمه C ولا توجد به قيمة ابتدائية .

هـ-متغير كلمة word اسمه word2 به قيمة ابتدائية 65536 .

و-مصفوفة كلمات اسمها Array1 وضع فيها قيمة ابتدائية

ز-ثابت اسمه Bell يساوي 7 .

حـ- ثابت رسالة اسمه msg يساوي 'This Is A Message $'

4-افترض أن البيانات التالية مخزنة في الذاكرة ابتداءً من الإزاحة 0000h

A DB 7
B DW 1ABCH
C DB 'HELLO'

أ-أعطِ عنوان الإزاحة للمتغيرات A,B,C .

ب- وضح محتويات البايت عند الإزاحة 0002h .

جـ- وضح محتويات البايت عند الإزاحة 0004h .

د- وضح عنوان الإزاحة للحرف 'O' في كلمة 'HELLO' .

5- وضح إذا كانت العبارات التالية صحيحة أو خطأً حيث B1,B2 عبارة عن متغيرات حرفية Byte وw1,w2 عبارة متغيرات كلمات words .

1-MOV Ds,Ax
2-MOV Ds,1000h
3- MOV CS,ES
4-MOV w1,DS
5-XCHG w1,w2
6-SUB 5,B1
7-ADD B1,B2
8-ADD AL,256
9-MOV w1,B1

6-استخدم الأوامر MOV, ADD , SUB ,INC , DEC , NEG لترجمة العبارات التالية المكتوبة بلغة راقية إلى عبارات بلغة التجميع :

1- A=B - A
2- A= -(A+1)
3- C= A + B
4- B= 3* B + 7
5- A= B - A- 1

7- اكتب عبارات (وليس برنامج كامل) لتقوم بالآتي :

1-قراءة حرف ثم طباعته في الموضع التالي في الشاشة في نفس السطر .

2- قراءة حرف كبير Upper case letter ثم طباعته في الموضع التالي بنفس السطر في الشاشة وذلك في صورة حرف صغير Lower case letter .

<u>برامج للكتابة :</u>

8- اكتب برنامج يقوم بالآتي :

1- طباعة العلامة '?' .

2- يقوم بقراءة رقمين عشريين مجموعهما أقل من العدد 10

3-يقوم البرنامج بحساب مجموع العددين وطباعة النتيجة في السطر التالي .

مثال للتنفيذ

? 35
The sum of 3 and 5 is 8

9- اكتب برنامج يقوم بطلب كتابة ثلاثة حروف . يقوم البرنامج بقراءة الحروف الثلاثة وطباعتها كل حرف في سطر منفصل .مثال للتنفيذ

Enter Three Letters :
ABC
A
B
C

10-اكتب برنامج يقوم بقراءة أحد الحروف في النظام السداسي عشر (A-F) يقوم البرنامج بطباعة الرقم المناظر في النظام العشري في السطر التالي . مثال للتنفيذ

Enter A Hexadecimal Digit :
C
In Decimal It Is : 12

الفصل الرابع

مسجل البيارق

أحد أهم مميزات الحاسب هي القدرة علي اتخاذ القرارات ويتم ذلك عن طريق تحديد حالة المعالج الدقيق بعد تنفيذ عملية محددة. في المعالج 8086 يتم تمثيل حالة المعالج بعد

تنفيذ آخر عملية في 9 خانات ثنائية تسمي البيارق Flags ويتم اتخاذ القرارات المختلفة حسب قيمة هذه البيارق. يتم تخزين البيارق في مسجل يسمي مسجل البيارق Flag Register ويمكن تقسيم البيارق إلي نوعين وهما بيارق التحكم Control Flags وبيارق الحالة Status Flags. وتقوم بيارق التحكم لتشغيل أو تعطيل عمليات محددة أثناء تنفيذ البرنامج بينما تقوم بيارق الحالة بعكس حالة المعالج بعد تنفيذ أمر محدد كأن يتم إظهار أن النتيجة تساوي صفر وذلك عن طريق رفع بيرق الصفر كما سنري في الجزء التالي.

مسجل البيارق

يحتوي هذا المسجل علي البيارق المختلفة كما هو موضح بالشكل حيث يتم تمثيل بيارق الحالة في الخانات 0 و 2 و 4 و 6 و 7 و 11 بينما تشغل بيارق التحكم الخانات 8 و 9 و 10 وتبقي بقية الخانات بدون استخدام (ليس من الضروري معرفة موقع البيرق من المسجل في أغلب الحالات حيث توجد أوامر للتخاطب مع كل بيرق علي حدة)، سنتناول في الجزء التالي بيارق الحالة

				Of	Df	If	Tf	Sf	Zf		Af		Pf		Cf

شكل يوضح مسجل البيارق

بيارق الحالة Status Flags

تقوم هذه البيارق بإظهار حالة المعالج بعد تنفيذ آخر أمر فمثلاً عند تنفيذ الأمر SUB Ax,Bx فإن بيرق الصفر يتأثر وتصبح قيمته تساوي 1 إذا كانت النتيجة تساوي صفر.

الجدول التالي يوضح البيارق المختلفة

بيارق الحالة Status Flags

الخانة	Name	الاسم	الرمز
0	Carry Flag	بيرق المحمول	CF
2	Parity Flag	بيرق خانة التطابق	PF
4	Auxiliary Carry Flag	بيرق المحمول المساعد	AF

6	Zero Flag	بيرق الصفر	ZF
7	Sign Flag	بيرق الإشارة	SF
11	Overflow Flag	بيرق الفيضان	OF
Control Flags بيارق التحكم			
8	Trap Flag	بيرق التنفيذ خطوة بخطوة	TF
9	Interrupt Flag	بيرق المقطعات	IF
10	Direction Flag	بيرق الاتجاه	DF

بيرق المحمول Carry Flag (CF)

يحتوي هذا البيرق علي القيمة '1' (يتم رفع البيرق) إذا وجد محمول من أو إلي الخانة ذات الوزن الأكبر Most Significant Bit (MSB) ويتم ذلك في حالات الجمع والطرح المختلفة. خلاف ذلك تكون قيمة البيرق تساوي صفر.

يتأثر البيرق أيضاً في حالة عمليات الإزاحة Shift والدوران Rotate والتي سنتحدث عنها فيما بعد.

بيرق التطابق Parity Flag (PF)

يحتوي هذا البيرق علي القيمة '1' إذا كان الحرف الأصغر من النتيجة Low Byte يحتوي علي عدد زوجي من الخانات التي تحتوي علي الرقم '1'. ويساوي صفر إذا كان عدد الخانات التي تحتوي علي الرقم '1' فردي. فمثلاً إذا كانت نتيجة آخر عملية هو الرقم FFFEh فإن الحرف الأصغر يحتوي علي العدد FEH (1111 1110) وبالتالي فإن عدد الخانات التي تحتوي علي الرقم '1' هو 7 خانات (عدد فردي) وعلي هذا فإن قيمة البيرق تساوي '0' (PF = 0)

بيرق المحمول المساعد Auxiliary Carry Flag (AF)

يحتوي هذا البيرق علي القيمة '1' إذا كان هناك محمول من أو إلي الخانة الرابعة bit-3 ويتم استخدام هذا البيرق في حالة الكود Binary Coded Decimal (BCD).

بيرق الصفر Zero Flag (ZF)

يحتوي هذا البيرق علي القيمة '1' (ZF=1) إذا كانت النتيجة تساوي صفر

بيرق الإشارة Sign Flag (SF)

يحتوي هذا البيرق علي القيمة '1' إذا كانت الخانة ذات الوزن الأكبر MSB تساوي '1' حيث يعني هذا

أن النتيجة سالبة. (أي أن SF = 1 إذا كانت MSB =1 و SF= 0 إذا كانت MSB= 0)

بيرق الفيضان Overflow Flag (OF)

يحتوي هذا البيرق علي القيمة '1' (OF=1) إذا حدث فائض في حالة الأرقام ذات الإشارة Signed Numbers وإلا فإنه سيحتوي علي صفر. وسنناقش هذا الموضوع بالتفصيل في الأجزاء المتبقية من هذا الفصل.

الفيضان Overflow

كما نعلم فإن إمكانية تخزين الأرقام في الحاسوب محدودة وذلك حسب المكان الذي سيتم فيه تخزين الرقم (مثلاً أكبر رقم يمكن تمثيله وتخزينه في خانة واحدة One Byte هو الرقم 255) وعلي ذلك إذا أردنا إجراء عملية حسابية وزاد الناتج عن هذه القيمة فإن المكان لن يسمح بتخزين النتيجة وفي هذه الحالة يكون قد حدث فيضان.

أمثلة علي الفيضان

يختلف الفيضان عند التحدث عن الأرقام الموجبة فقط (الأرقام بدون إشارة) Unsigned Numbers عنه في الأرقام بإشارة Signed Numbers. وعند إجراء عملية مثل الجمع هنالك أربع احتمالات للنتيجة:

1 - لا يوجد فيضان

2 - فيضان بإشارة فقط

3 - فيضان بدون إشارة فقط

4 - فيضان بإشارة وبدون إشارة

وكمثال للفيضان بدون إشارة وليس بإشارة افترض أن المسجل AX يحتوي علي الرقم FFFFh وأن المسجل BX يحتوي علي الرقم 1 وقمنا بتنفيذ الأمر ADD AX, BX ستكون النتيجة علي النحو التالي :

	1111	1111	1111	1111
+	0000	0000	0000	0001
=	1	0000	0000	0000
			0000	

وبالتالي يكو لدينا أحد احتمالين

1- إذا فسرنا هذه الأرقام علي أنها أرقام بدون إشارة فإن النتيجة الصحيحة هي الرقم 65536 أي الرقم السداسي عشر 10000h ولكن هذه النتيجة لا يمكن تخزينها في المسجل (أكبر من أكبر رقم يمكن تخزينه 65535) حيث سيتم فقد الرقم 1 وتخزين الرقم 0000h في المسجل AX وبالتالي فإن النتيجة التي تم تسجيلها هي نتيجة خاطئة.

2- أما إذا فسرنا هذه الأرقام علي أنها أرقام بإشارة فإن الرقم الأول FFFFh هو الرقم 1- وعند جمع الرقم 1 إليه فإن النتيجة هي الرقم 0 وعلي هذا فإن النتيجة التي تم تخزينها (الرقم 0) صحيحة وعلي هذا لم يحدث فيضان بإشارة.

مثال آخر لفيضان بإشارة وليس بدون إشارة، افترض أن كل من المسجلين AX و BX يحتويان علي العدد 7FFFh وتم تنفيذ الأمر ADD AX,BX تكون النتيجة علي النحو التالي:

```
     0111   1111   1111   1111
 +   0111   1111   1111   1111
 --------------------------------
     1111   1111   1111   1110
                      =   FFFEh
```

وفي هذه الحالة التفسير للرقم 7FFFh في حالة الأرقام بإشارة أو بدون إشارة هو تفسير واحد حيث أن الخانة ذات الوزن الأكبر تساوي 0 (MSB = 0) وهو الرقم 32767 (7FFFh) وعلي ذلك فإن نتيجة حاصل الجمع يجب أن تكون واحدة في الحالتين وهي الرقم 65534 وهذه النتيجة لا يمكن تخزينها في حالة الأرقام بإشارة حيث أن تفسير هذه النتيجة في حالة الأرقام بإشارة هو الرقم السالب (2-)

وعلي ذلك فلدينا في هذا المثال فيضان بإشارة ولا يوجد فيضان بدون إشارة

كيف يقوم المعالج بتوضيح حدوث الفيضان ؟

يقوم المعالج برفع بيرق الفيضان OF=1 إذا حدث فيضان بإشارة ورفع بيرق المحمول إذا حدث فيضان بدون إشارة CF=1

وتصبح وظيفة البرنامج التأكد من حدوث أي من أنواع الفيضانات التي ذكرناها واتخاذ الإجراءات المناسبة. وإذا تم تجاهل هذه البيارق وحدث فيضان فقد تكون النتيجة غير صحيحة.

وعلي هذا فإن المعالج لا يفرق بين الأرقام بإشارة أو بدون إشارة فهو فقط يقوم برفع البيارق لبيان حدوث أي من الفيضان بإشارة أو بدون إشارة. فإذا كنا في البرنامج نتعامل مع الأرقام علي أنها بدون إشارة فإننا نهتم ببيرق المحمول فقط CF ونتجاهل بيرق الفيضان OF. أما إذا كنا نتعامل مع الأرقام بإشارة فإن بيرق الفيضان OF هو الذي يهمنا.

كيف يقوم المعالج بتحديد حدوث الفيضان ؟

كثير من الأوامر تؤدي إلي حدوث فيضان وسنناقش هنا أوامر الجمع والطرح للتبسيط

الفيضان بدون إشارة Unsigned overflow

في حالة الجمع يحدث فيضان بدون إشارة إذا كان هناك محمول من الخانة ذات الوزن الأكبر MSB حيث يعني هذا أن النتيجة أكبر من أن يتم تخزينها في المسجل المستودع (أي أن النتيجة أكبر من أكبر رقم يمكن تخزينه وهو الرقم FFFFh في حالة أن يكون المستودع به 16 خانة ثنائية أو FFh في حالة أن يكون المستودع به 8 خانات ثنائية).

في حالة الطرح يحدث الفيضان في حالة الاستلاف للخانة ذات الوزن الأكبر حيث يعني هذا ان النتيجة أقل من الصفر (رقم سالب).

الفيضان بإشارة Signed Overflow

في حالة جمع أرقام بنفس الإشارة يحدث الفيضان في حالة أن تكون إشارة حاصل الجمع مختلفة عن إشارة الرقمين. كما نجد أنه في حالة طرح رقمين بإشارة

مختلفة فإن العملية تشابه عملية الجمع لرقمين بإشارة واحدة حيث أن

$$A-(-B)=A+B, \quad -A-(+B)=-A-B$$

ويحدث الفيضان بإشارة إذا اختلفت إشارة النتيجة عن الإشارة المتوقعة كما في حالة عملية الجمع

أما في حالة جمع رقمين بإشارتين مختلفتين فإن حدوث الفيضان مستحيل حيث أن العملية (B)-+A هي عبارة عن A-B وحيث أن الأرقام A و B أرقام صغيرة أمكن تمثيلها فإن الفرق بينهما هو أيضاً رقم صغير يمكن تمثيله . وبالمثل فإن عملية الطرح لرقمين بإشارتين مختلفتين لن تعطي أي فيضان.

وعموماً فإن المعالج يقوم برفع بيرق الفيضان كالآتي :

إذا كان المحمول إلي الخانة ذات الوزن الأكبر MSB والمحمول من الخانة ذات الوزن الأكبر مختلفان (ويعني هذا أنه يوجد محمول إليها ولا يوجد محمول منها أو لا يوجد محمول إليها ولكن يوجد محمول منها). في هذه الحالة يتم رفع بيرق الفيضان (أنظر الأمثلة لاحقاً).

كيف تؤثر العمليات علي البيارق:

عندما يقوم المعالج بتنفيذ أي أمر يتم رفع البيارق المناسبة لتوضيح النتيجة . وعموماً هنالك أوامر لا تؤثر في كل البيارق وإنما تؤثر في بعضها فقط إذ قد تترك كل البيارق دون تأثير . وعموماً فإن عملية تفرع البرنامج باستخدام أوامر التفرع JUMP INSTRUCTIONS تعتمد عملياً علي قيم البيارق المختلفة كما سنري فيما بعد .

في هذا الجزء سنوضح تأثير البيارق في حالة تنفيذ بعض الأوامر التي ناقشناها وتعاملنا معها في الفصل السابق :

الأمر	البيارق المتأثرة
MOV / XCHG	لا تتأثر أي من البيارق
ADD / SUB	تتأثر كل البيارق
INC / DEC	تتأثر كل البيارق عدا بيرق المحمول (CF)
NEG	تتأثر البيارق (CF=1 إلا إذا كانت النتيجة تساوي 0 ، OF=1 إذا كان المعامل هو

الرقم 800H في حالة WORD أو 80h في حالة المعامل Byte)	

لتوضيح تأثر البيارق بتنفيذ العمليات سنقوم بعمل بعض الأمثلة في كل مثال سنوضح الأمر ومحتوي المعاملات operands وحساب وتوقع قيم البيارق المختلفة 0f,sf,zf,pf,cf (سنتجاهل بيرق المحمول المساعد AF لأنه في الحالة ذات الأرقام من النوع BCD فقط).

مثال 1:

نفذ الأمر ADD AX,BX حيث يحتوي المسجل AX علي الرقم FFFFh والمسجل BX علي الرقم FFFFh

الحل :

```
 FFFFh
+FFFFh
1FFFEh
```

يتم تخزين الرقم 1110 1111 1111 1111 (0FFFEh) في المسجل AX وعلي هذا تكون البيارق علي النحو التالي:

بيرق الإشارة SF : يساوي 1 لأن قيمة الخانة ذات الوزن الأعلى MSB تساوي 1 .

بيرق خانة التطابق PF : يساوي 0 لأن لدينا عدد 7 خانات (عدد فردي) تحتوي علي 1 في النصف الأدنى

LOW BYTE في النتيجة .

بيرق الصفر ZF : يساوي 0 لأن النتيجة لا تساوي صفر .

بيرق المحمول CF : يساوي 1 لأن هناك محمول في الخانة ذات الوزن الأكبر MSB في عملية الجمع .

بيرق الفيضان OF : يساوي صفر لأن إشارة النتيجة هي نفس إشارة الأرقام التي تم جمعها

(المحمول إلي الخانة MSB لا يختلف عن المحمول من الخانة MSB).

<u>مثال 2 :</u>

نفذ الأمر ADD AL,BL حيث يحتوي AL علي الرقم 80h و BL علي الرقم 80h

الحل :

$$
\begin{array}{r}
80h \\
\underline{+80h} \\
100h
\end{array}
$$

يحتوي المسجل AL علي الرقم 00h

بيرق الإشارة SF : SF=0 لأن الخانة MSB تحتوي علي 0

بيرق خانة التطابق PF: PF=1 لأنه لدينا عدد 0 خانة تحتوي علي الرقم 1 ويعتبر الصفر عدد زوجي

بيرق الصفر ZF : ZF=1 لأن النتيجة تساوي 0

بيرق المحمول CF : CF=1 لأن هناك محمول من الخانة ذات الوزن الأكبر MSB

بيرق الفيضان OF : OF=1 لأن الأرقام المجموعة سالبة بينما النتيجة موجبة (المحمول إلي الخانة MSB لا يساوي المحمول منها).

مثال 3:

نفذ الأمر SUB AX,BX إذا كان المسجل AX يحتوي علي الرقم 8000h والمسجل BX يحتوي علي الرقم 0001h

الحل :

8000h

-0001h

7FFFh = 0111 1111 1111 1111

بيرق الإشارة SF : SF=0 لأن خانة MSB=0

بيرق خانة التطابق PF: PF=1 لأن الخانة الصغري من النتيجة بها 8 خانات (عدد زوجي) بها "1"

بيرق الصفر ZF : ZF=0 لأن النتيجة لاتساوي 0

بيرق المحمول CF : CF=0 لأننا قمنا بطرح عدد صغير بدون إشارة من عدد أكبر منه

بيرق الفيضان OF : OF=1 في حالة الأرقام بإشارة فإننا نطرح رقم موجب من رقم سالب . وهي مثل عملية جمع رقمين سالبين. ولأن النتيجة أصبحت موجبة (إشارة النتيجة خطأ) .

مثال 4:

نفذ الأمر INC AL حيث AL يحتوي علي الرقم FFh

الحل :

```
 FFh
+  1h
-----
 100h
```

يتم تخزين الرقم 100h في المسجل AL .بعد تنفيذ هذه العملية نجد أن

بيرق التطابق : 1 (PF=1) .

بيرق الإشارة SF : SF=0 لأن MSB=0

بيرق خانة التطابق PF: PF=1 لوجود 8 خانات تحتوي علي "1" في البايت الأدنى من النتيجة

بيرق الصفر ZF : ZF=1 لأن النتيجة تساوي صفر

بيرق المحمول CF : لا يتأثر بالأمر INC بالرغم من حدوث فائض.

بيرق الفيضان OF : OF=0 وذلك لأننا نجمع رقم سالب إلي رقم موجب (المحمول إلي الخانة MSB يساوي المحمول منها).

__مثال 5:__

نفذ الأمر MOV AX,-5

يتم وضع الرقم 5– (FFFBh) في المسجل AX ولا تتأثر أي من البيارق بالأمر MOV .

__مثال 6:__

نفذ الأمر NEG AX حيث يحتوي المسجل AX علي الرقم 8000h

8000h = 1000 0000 0000 0000

COMPLEMENT = 1000 0000 0000 0000

1000 0000 0000 0000

بيرق الإشارة SF : SF=1

بيرق خانة التطابق PF: PF=1

بيرق الصفر ZF : ZF=0

بيرق المحمول CF : CF=1 لأنه في حالة تغيير الإشارة فإن CF =1 دائماً إلا إذا كان الرقم يساوي صفر .

بيرق الفيضان OF : OF=1 لأننا عند تنفيذ الأمر NEG نتوقع تغيير إشارته وفي هذه الحالة لم تتغير الإشارة .

برنامج DEBUG :

يمكن باستخدام برنامج DEBUG متابعة تنفيذ البرنامج خطوة_خطوة وإظهار النتيجة وتأثر المسجلات بعد كل خطوة كما يمكن كتابة برنامج بلغة التجميع حيث يقوم بتحويله إلي لغة الآلة مباشرة وتخزينها في الذاكرة

ولاستعمال برنامج الـ DEBUG نقوم بكتابة برنامج بلغة التجميع وتجهيزه حتى نحصل علي الملف القابل للتنفيذ EXCUTABLE FILE بعد ذلك يمكننا تحميل البرنامج بواسطة الأمر

C:\DOS\DEBUG TEST.EXE

يقوم البرنامج بالرد بالإشارة "-" دليل علي أنه في حالة انتظار لأحد الأوامر وهنا توضيح لبعض الأوامر الهامة :-

1. الأمر R وهو يوضح محتويات المسجلات . ولوضع قيمة محددة في أحد المسجلات يتم كتابة الأمر R متبوعاً بإسم المسجل (مثلاً R IP).
2. الأمر T (TRACE) وهو يؤدي إلي تنفيذ الخطوة الحالية فقط من البرنامج .

3. الأمر G (GO) يؤدي إلي تنفيذ البرنامج .
4. الأمر Q (QUIT) يؤدي إلي الخروج من البرنامج .
5. الأمر ASSEMBLE A يتيح فرصة كتابة برنامج .
6. الأمر U لرؤية جزء من الذاكرة .
7. الأمر DUMB D يؤدي إلي إظهار جزء من الذاكرة .

لتجربة برنامج Debug دعنا نتابع تنفيذ البرنامج التالي:

```
MODEL SMALL
.STACK 100H
.CODE
MAIN    PROC
MOV     AX , 4000H        ;ax =
4000h
ADD     AX , AX       ;ax = 8000h
SUB     AX , 0FFFFH       ;ax =
8001h
NEG     AX            ;ax = 7fffh
INC AX            ;ax = 8000h
MOV     AH , 4CH
INT 21H           ;DOS exit
MAIN    ENDP
END     MAIN
```

بعد كتابة البرنامج السابق وليكن اسمه test.asm وتوليد الملف القابل للتنفيذ Executable file والذي سيحمل الاسم Test.exe يتم نداء برنامج Debug وتحميل البرنامج وذلك بتنفيذ الأمر التالي من محث الـ DOS: c:\asm> DEBUG TEST.EXE

يقوم البرنامج بالتحميل وإظهار المؤشر "-" والذي تشير للاستعداد لتلقي الأوامر.

نبدأ بتجربة الأمر R وذلك لإظهار محتويات المسجلات المختلفة وتكون المخرجات على الصورة التالية:

```
- R
AX=0000  BX=0000  CX=001F  DX=0000  SP=000A  BP=0000  SI=0000  DI=0000
DS=0ED5  ES=0ED5  SS=0EE5  CS=EE6  IP=0000  NV  UP  DI  PL  NZ  NA  PO  NC
0EE6:0000          B80040  MOV  AX , 4000
```

يقوم البرنامج بإظهار محتويات المسجلات المختلفة وفي السطر الثالث يوضع عنوان الأمر التالي (المطلوب تنفيذه - لاحظ قيمة العنوان ومحتويات المسجلين CS:IP) متبوعاً بكود الآلة للأمر Machine Code وهو الرقم B80040 وبعد ذلك نجد الأمر مكتوباً بلغة التجميع.

عند تشغيل البرنامج ستجد أرقام مختلفة عن الأرقام الموضحة في هذا المثال وبالذات محتويات المسجلات المختلفة.

في نهاية السطر الثاني يوجد عدد 8 أزواج حروف عل الصورة NV UP DI PL NZ NA PO NC توضح محتويات البيارق المختلفة وذلك حسب الجدول التالي:

البيرق	في حالة رفع البيرق Set	في حالة عدم رفع البيرق Clear
CF (CarryFlag)	CY (CarrY)	NC (No Carry)
PF (Parity Flag)	PE (Parity Even)	PO (Parity Odd)
AF (Auxiliary Flag)	AC (Auxiliary Carry)	NA (No Auxiliary carry)
ZF (Zero Flag)	ZR (ZeRo)	NZ (NonZero)
SF (Sign Flag)	NG (NeGative)	PL (Plus)
OF (Overflow Flag)	OV (OVerflow)	NV (No oVerflow)
بيارق التحكم Control Flags		
DF (Direction Flag)	DN (DownN)	UP (UP)
IF (Interrupt Flag)	EI (Enable Interrupt)	DI (Disable Interrupt)

لبداية تشغيل البرنامج نصدر الأمر T أي Trace للتنفيذ خطوة خطوة فيكون التسلسل التالي للأوامر :

في البداية كانت المسجلات علي النحو التالي (سنكرر الشاشة السابقة حتى نتابع التنفيذ بالتفصيل

```
- R
AX=0000  BX=0000  CX=001F  DX=0000  SP=000A  BP=0000  SI=0000  DI=0000
DS=0ED5  ES=0ED5  SS=0EE5  CS=EE6  IP=0000  NV  UP  DI  PL  NZ  NA  PO  NC
0EE6:0000          B80040  MOV  AX , 4000
```

ثم نبدأ التنفيذ: الأمر الأول MOV AX , 4000h

```
- T
AX=4000  BX=0000  CX=001F  DX=0000  SP=000A  BP=0000  SI=0000  DI=0000
DS=0ED5  ES=0ED5  SS=0EE5  CS=EE6  IP=0003  NV  UP  DI  PL  NZ  NA  PO  NC
0EE6:0003          03C0     ADD  AX , AX
```

التنفيذ يضع 4000h في المسجل AX

لاحظ أن المسجل AX أصبح به الرقم 4000H ولم يتم تغيير محتويات البيارق وأن الأمر التالي أصبح الأمر ADD AX,AX

```
- T
AX=8000  BX=0000  CX=001F  DX=0000  SP=000A  BP=0000  SI=0000  DI=0000
DS=0ED5  ES=0ED5  SS=0EE5  CS=EE6  IP=0005  OV  UP  DI  NG  NZ  NA  PE  NC
0EE6:0005          2DFFFF SUB  AX , FFFF
```

لاحظ أن المسجل AX أصبح به الرقم 8000H وأن النتيجة السابقة أثرت في البيارق حيث تم رفع بيرق الفيضان ليشير إلي حدوث فيضان بإشارة وبيرق الإشارة ليشير إلي أن النتيجة سالبة وكذلك بيرق التطابق لأن الخانة الأصغر من المسجل AX (أي AL) تحتوي علي عدد زوجي من الخانات التي بها الرقم 1 . والآن نتابع تنفيذ البرنامج حيث الأمر التالي هو الأمر SUB AX,FFFFh

```
- T
AX=8001  BX=0000  CX=001F  DX=0000  SP=000A  BP=0000  SI=0000  DI=0000
DS=0ED5  ES=0ED5  SS=0EE5  CS=EE6  IP=0008  NV  UP  DI  NG  NZ  AC  PO  CY
0EE6:0008        F7D8    NEG  AX
```

```
- T
AX=7FFF  BX=0000  CX=001F  DX=0000  SP=000A  BP=0000  SI=0000  DI=0000
DS=0ED5  ES=0ED5  SS=0EE5  CS=EE6  IP=000A  NV  UP  DI  PL  NZ  AC  PE  CY
0EE6:000A        40        INC   AX
```

```
-T
AX=8000  BX=0000  CX=001F  DX=0000  SP=000A  BP=0000  SI=0000  DI=0000
DS=0ED5  ES=0ED5  SS=0EE5  CS=EE6  IP=000B  OV  UP  DI  NG  NZ  AC  PE  CY
0EE6:000B        B44C     MOV  AH , 4C
```

```
- G
PROGRAM TERMINATED NORMALLY
-Q
C:\>
```

تمارين :

وضع محتويات المسجل المستودع DESTINATION REG

وكذلك قيم البيارق بعد تنفيذ كل من الأوامر التالية .

1. ADD AX,BX حيث يحتوي المسجل AX علي الرقم 7FFFh والمسجل BX علي 0001h.

2. SUB AL,B حيث AL=01h و BL=FFh

3. DEC AL حيث AL=00h

4. NEG AL حيث AL=7F

5. XCHG AX,BX حيث AX=1ABCh و BX=712h .

6. ADD AL,BL حيث AL=80h و BL=FFh .

7. SUB AX,BX حيث AX=0000h و BX=8000h .

8. NEG AX حيث AX=0001h .

2-أفترض ان المسجلين AX BX يحتويان علي أرقام موجبة . وتم تنفيذ الأمر ADD AX,BX وضح أنه يوجد

محمول إلي الخانة MSB ولا يوجد محمول منها وذلك فقط في حالة حدوث فيضان بإشارة .

أفترض ان المسجلين AX BX يحتويان علي أرقام سالبة . وتم تنفيذ الأمر ADD AX,BX وضع أنه يوجد محمول من الخانة MSB ولا يوجد محمول إليها وذلك فقط في حالة حدوث فيضان بإشارة .

3- أفترض أن الأمر ADD AX,BX تم تنفيذه إذا كانت محتويات المسجل AX هي الرقم الأول بينما المسجل BX به الرقم التالي . وضع محتويات المسجل AX في كل من الحالات الآتية موضحاً حدوث فيضان بإشارة أو بدون إشارة .

أ. 512Ch   ب. FE12h   ج. E1E4h   د. 7132h
هـ. 6389h

+ 4185h   +1ACBh   +DAB3h   +7000h
+1176h

4- أفترض أن الأمر SUB AX,BX تم تنفيذه إذا كانت محتويات المسجل AX هي الرقم الأول بينما المسجل BX به الرقم التالي . وضع محتويات المسجل AX في كل من الحالات الآتية موضحاً حدوث فيضان بإشارة أو بدون إشارة .

أ. 2143h ب. 81Feh ج. 19BCh د.

0002h هـ 88CDh

-1986h -1986h -81Feh

FE0Fh 71ABh

الفصل الخامس

التفرع وتعليمات ضبط الانسياب Flow Control Instructions

لكي نكتب برنامج يقوم بعمل محدد غالبا ما يتم استخدام أوامر التفرع التي تجعل المبرمج قادراً علي اتخاذ قرارات محددة وتؤدي أوامر التفرع والتكرار إلي تنفيذ برامج فرعية ويعتمد هذا التفرع أو التكرار عادة علي قيم محددة للمسجلات وذلك عن طريق بيارق الحالة Status Flags والتي تتأثر دائماً بآخر عملية تم تنفيذها.

سنقوم في هذا الفصل بتوضيح أوامر التفرع المختلفة وسنستخدمها في تمثيل عبارات التكرار والتفرع في اللغات العليا HIGH LEVEL LANGUAGE وذلك بإعادة كتابتها بلغة التجميع .

مثال للتفرع :

لتوضيح عمل أوامر التفرع سنبدأ بمثال يقوم بطباعة الحروف المستخدمة كلها وذلك عن طريق طباعة جدول الحروف ASCII Table كاملاً .

```
.Model    Small
.Stack    100h
.Code
MAINPROC
MOV    AH , 2
MOV    CX , 256
MOV    DL , 0
Print_Loop:
INT 21h          ;DL اطبع الحرف الموجود في المسجل
INC DL           ; تجهيز الحرف التالي
DEC    CX        ; انقص العداد
JNZ PRINT_LOOP   ; إذا لم ننتهي تفرع إلى العنوان المحدد
; DOS_EXIT
MOV    AH , 4Ch
INT 21h
MAINENDP
END  MAIN
```

يوجد لدينا عدد 256 حرف في IBM Character Set منها الحروف والأرقام والحروف الخاصة. لإظهار الحروف في الشاشة يتم استخدام الخدمة رقم 2 (إظهار حرف واحد فقط) وذلك بوضع الرقم 2 في المسجل AH. تم استخدام المسجل DL ليحوى الحرف المطلوب طباعته لذلك تم وضع الرقم 0 فيه كقيمة ابتدائية وزيادته في كل مرة كما تم استخدام المسجل CX كعداد بقيمة ابتدائية 256 وإنقاصه في كل مرة حتى تصل قيمته إلي الصفر.

استخدم الأمر JNZ (Jump if Not Zero) وهو الأمر الذي يضبط الحلقة وذلك للتفرع إلى العنوان المحدد (Print-Loop) إذا تم إنقاص المسجل CX بواحد ولم تصل النتيجة إلى الصفر ويتم ذلك عن طريق استعمال بيرق الصفر ZF . فإذا كانت النتيجة لا تساوي صفر

(ZF= 0) يتم القفز إلى العنوان المحدد أما إذا كانت النتيجة تساوي الصفر (ZF= 1) يتم الاستمرار في البرنامج و العودة إلى نظام التشغيل باستخدام الخدمة رقم 4CH.

التفرع المشروط CONDITIONAL JUMP

الأمر JNZالسابق هو مثال لأوامر التفرع المشروط. و يكون أمر التفرع المشروط على الصورة

```
Jxxx      destination-Label
```

فإذا تحقق الشرط المحدد يتم تفرع البرنامج إلى العنوان الموضح كمعامل للأمر، ويكون الأمر التالي هو الأمر الموجود في العنوان المحدد .أما إذا لم يتحقق الشرط يتم الاستمرار كالمعتاد إلى الأمر التالي مباشرة .

في حالة التفرع يجب أن يكون العنوان الذي سيتم التفرع عليه على بعد 126 قبل العنوان الحالي أو 127 بعد العنوان الحالي وسنرى فيما بعد كيفية التفرع إلى أماكن أبعد من هذا المدى .

كيف يقوم المعالج بتنفيذ عملية التفرع المشروط ؟

يقوم المعالج باستخدام البيارق لتحديد عملية التفرع . حيث أن البيارق تعكس الحالة بعد تنفيذ آخر عملية وبالتالي فإن أوامر التفرع يجب أن تعتمد على بيرق محدد أو بيارق محددة حيث يتم التفرع إذا تم رفع هذه البيارق .

إذا تحقق التفرع يقوم المعالج بتحميل مؤشر التعليمات IP بالقيمة المحددة بالعنوان الموجود في أمر التفرع .أما إذا لم يتم تحقق الشرط فإن مؤشر التعليمات يواصل إلى العنوان التالي مباشرة .

ففي المثال السابق نجد الأمر

```
JNZ    PRINT-LOOP
```

وهذا يعني أنه إذا كان بيرق الصفر لا يساوي واحد ZF= 0 فإنه يتم التفرع إلى العنوان PRINT-LOOP وذلك بتحميل مؤشر

التعليمات بالعنوان . أما إذا كانت النتيجة تساوي الصفر (ZF= 1)
فإن البرنامج يواصل إلى الخطوة التالية.

تنقسم أوامر التفرع المشروط إلى ثلاثة مجموعات :

- المجموعة الأولى التفرع بالإشارة Signed Jumps
وتستخدم في حالة استخدام الأرقام بالإشارة Singed Numbers
- المجموعة الثانية التفرع بدون إشارة Unsigned Jumps
وتستخدم في حالة استخدام الأرقام بدون إشارة Unsigned Numbers .
- التفرع ببيرق واحد Single Flag Jumps والتي تعتمد على بيرق محدد .

الجداول التالية توضح أوامر التفرع المختلفة . لاحظ أن الأمر قد يأخذ أكثر من اسم مثلا JG و JNLE حيث تعني تفرع إذا كانت النتيجة أكبر JG أو تفرع إذا كانت النتيجة ليست أصغر من أو تساوي . ويمكن استخدام أي من الأمرين لأنهما يؤديان إلى نفس النتيجة .

1- التفرع بالإشارة Signed Jumps

الأمر	الوصف	شرط التفرع
JG / JNLE	تفرع في حالة أكبر من (ليس أصغر من أو يساوي)	ZF=0 & SF=OF
JGE / JNL	تفرع في حالة أكبر من أو يساوي (ليس أصغر من)	SF=OF
JL / JNGE	تفرع في حالة أقل من (ليس أكبر من أو يساوي)	SF<>OF
JLE / JNG	تفرع في حالة أقل من أو يساوي (ليس أكبر من)	ZF=1 OR SF<>OF

2- التفرع بدون إشارة Unsigned Jumps

الأمر	الوصف	شرط التفرع
JA / JNBE	تفرع في حالة أكبر من (ليس أصغر من أو يساوي)	CF=0 & ZF=0
JAE / JNB	تفرع في حالة أكبر من أو يساوي (ليس أصغر من)	CF=0
JB / JNAE	تفرع في حالة أقل من (ليس أكبر من أو يساوي)	CF=1

CF=1 OR ZF=1	تفرع في حالة أقل من أو يساوي (ليس أكبر من)	JBE / JNA

3-التفرع ببيرق واحد Single Flag Jumps

شرط التفرع	الوصف	الأمر
ZF=1	تفرع في حالة التساوي أو الصفر	JE / JZ
ZF=0	تفرع في حالة عدم التساوي (لا يساوي الصفر)	JNE / JNZ
CF=1	تفرع في حالة محمول Carry	JC
CF=0	تفرع في حالة عدم وجود محمول Carry	JNC
OF=1	تفرع في حالة الفيضان	JO
OF=0	تفرع في حالة عدم حدوث الفيضان	JNO
SF=1	تفرع في حالة النتيجة سالبة	JS
SF=0	تفرع في حالة النتيجة موجبة	JNS
PF=1	تفرع في حالة التطابق الزوجي	JP / JPE
PF=0	تفرع في حالة التطابق الفردي	JNP / JPO

الأمر CMP

الأمر Compare(CMP) يستخدم لمقارنة رقمين ويأخذ

الصيغة :

CMP Destination , Source

يقوم البرنامج بعملية المقارنة عن طريق طرح المصدر source من المستودع destination ولا يتم تخزين النتيجة ولكن البيارق تتأثر ، لا يقوم الأمر CMP بمقارنة موضعين في الذاكرة كما أن المستودع destination لا يمكن أن يكون رقم ثابت .

لاحظ أن الأمر CMP يماثل تماما الأمر SUB فيما عدا أن النتيجة لا يتم تخزينها .

افترض أن البرنامج يحتوي على التالي:

```
CMP    Ax , Bx
JG     Below
```

حيث AX=777Fh، BX=0001h فان نتيجة الأمر Ax,Bx CMP هي:

7FFFh - 0001h = 7FFEh

والتفرع هنا يتم حيث أن البيارق تكون zf = sf = of =0

والأمر JG يتطلب أن تكون

Zf = 0 و كذلك Sf = Of وعلى هذا يتم التفرع إلى العنوان المحدد Below .

في حالة التفرع المشروط ورغم أن عملية التفرع تتم حسب حالة البيارق المختلفة فان المبرمج ينظر إلى الأمر بدون تفاصيل البيارق فمثلا :

```
CMP        AX,BX
JG         Below
```

إذا كان الرقم الموجود في المسجل AX أكبر من الرقم الموجود في المسجل BX فان البرنامج يتفرع إلى العنوان Below .

بالرغم من أن الأمر CMP صمم خصيصا للتعامل مع التفرع المشروط ولكن يمكن لعبارة التفرع المشروط أن تكون بعد أي أمر آخر مثلا :

```
DEC        CX
JNZ        loop
```

يتم هنا التفرع إلى العنوان loop إذا لم تكن قيمة المسجل CX تساوي صفر.

التفرع بإشارة والتفرع بدون إشارة:

كل أمر تفرع بإشارة يناظره أمر تفرع بدون إشارة ، مثلا الأمر JG يناظره الأمر JA واستخدام أي منهما يعتمد على طريقة التعامل مع الأرقام داخل البرنامج. حيث أن الجدول السابق قام بتوضيح أن كل عملية من هذه العمليات تعتمد على بيارق محددة حيث أن التفرع بإشارة يتعامل مع البيارق zf , sf , of بينما التفرع بدون إشارة يعتمد على البيارق zf , cf واستخدام الأمر غير المناسب قد يؤدي إلى نتائج غير صحيحة .

مثلا إذا استخدمنا الأرقام بإشارة وكان المسجل Ax يحتوي على الرقم 7fffh والمسجل Bx يحتوي على الرقم 8000h وتم تنفيذ الأوامر التالية :

```
CMP     AX,BX
JA   Below
```

فبالرغم من أن 7EFF > 8000h في حالة الأرقام بإشارة فان البرنامج لن يقوم بالتفرع إلى العنوان Below وذلك لأن 7FFFh < 8000h في حالة الأرقام بإشارة ونحن نستعمل الأمر JA الذي يتعامل مع الأرقام بدون إشارة .

التعامل مع الحروف:

عند التعامل مع الحروف يمكن استخدام الأرقام بإشارة أو بدون إشارة ذلك لأن الحروف تحتوي على الرقم 0 في الخانة ذات الوزن الأكبر MSB وعموما نستخدم الأرقام بدون إشارة في حالة التعامل مع الحروف المسماة الممتدة Extended ASCII Code والواقعة في المدى 80h - FFh .

مثال :

افترض أن المسجلين AX و BX يحتويان علي أرقام بإشارة، اكتب جزء من برنامج يضع القيمة الأكبر في المسجل CX.

```
MOV      CX , AX
CMP      BX , CX
JLE NEXT
MOV      CX,BX
NEXT:
```

التفرع الغير مشروط Unconditional Jump

يستخدم الأمر JMP للتفرع إلي عنوان محدد وذلك بدون أي شروط حيث الصيغة العامة للأمر هي:

```
Jmp      Destination
```

ويكون العنوان الذي سيتم التفرع إليه داخل مقطع البرنامج الحالي وعلي ذلك فإن المدى الذي يمكن التفرع إليه أكبر

من حالة التفرع المشروط. ويمكن استغلال هذه الخاصية كما في الجزء التالي وذلك لتحسين أداء التفرع المشروط.

TOP:

عبارات الحلقة ; Loop Body

انقص واحد من العداد ; Dec CX

استمر في التفرع إذا كان ; JNZ TOP

العداد لا يساوي صفر

إذا احتوت الحلقة علي عبارات كثيرة بحيث يكون العنوان TOPبعيد جداً (أبعد من 126 خانة) فإن الأمر JNZ لن يصلح ولكن يمكن علاج هذه المشكلة بإعادة كتابة البرنامج علي النحو التالي واستخدام الأمر JMP الذي يتيح لنا التعامل مع مدي أكبر

```
TOP:
; Loop Body عبارات الحلقة
DEC     CX
JNZ BOTTOM
JMP     EXIT
BOTTOM:
JMP     TOP
EXIT:
```

هيكلية البرنامج

ذكرنا أن عمليات التفرع يمكن استخدامها في التفرع والتكرار ولأن أوامر التفرع بسيطة سنتطرق في هذا الجزء لكيفية كتابة أوامر التكرار والتفرع والمستخدمة في لغات البرمجة الراقية High Level Programming Languages .

أوامر التفرع

الأمر IF……..Then…….

الشكل العام لعبارة If..Then… هو

IF condition is True then

Execute True branch statements

End_IF

أي إذا تحقق الشرط يتم تنفيذ الأوامر وإذا لم يتحقق لا يتم تنفيذ شيء

مثال استبدل محتويات المسجل AX بالقيمة المطلقة لها.

أي إذا كانت محتويات المسجل سالبه (اقل من صفر) استبدلها بالقيمة الموجبة.

IF AX < 0 then

Replace AX with –AX

End_IF

بلغة التجميع تصبح

```
CMP      AX , 0
JNL END_IF
; Then
NEG      AX
END_IF:
```

2 – عبارة IF…THEN…..ELSE…..ENDIF

وهي تكون علي الصورة

```
IF  Condition is True then
    Execute True_Branch statements
ELSE
    Execute False_Branch statements
End_IF
```

إذا تحقق الشرط يتم تنفيذ مجموعة من الأوامر وإذا لم يتحقق يتم تنفيذ مجموعة أخري من الأوامر

مثال:-

افترض أن BL,AL يحتويان حروف (ASCII CODE) ، قم بعرض الحرف الأول بالترتيب (ذو القيمة الأصغر)

```
IF    AL <=  BL  THEN
      DISPLAY    AL
            ELSE
      DISPLAY  BL
          END_IF
```

(تصبح بلغة التجميع) كالآتي :-

```
AH,2   MOV
CMP      AL,BL
JNBE     ELSE_
MOV      DL,AL
JMP      DISPLAY
ELSE_:
MOV      DL,BL
DISPLAY:
INT          21H
```

3- عبارة CASE

في حالة عبارة CASE يوجد اكثر من مسار يمكن ان يتبعه البرنامج والشكل العام للأمر هو :

```
CASE          EXPRESSION
VALUE_1    :
STATEMENT_1
VALUE_2    :    STATEMENT_2
:
VALUE_N    :    STATEMENT_N
```

END_CASE

مثال:

إذا كان المسجل AX يحتوى على رقم سالب ضع الرقم 1– في المسجل BX فإذا كان AX به صفر ضع الرقم 0 في المسجل BX أما إذا كان المسجل AX به رقم موجب ضع الرقم 1 في المسجل BX.

الحل:

```
CASE    AX
< 0  :    PUT    –1   IN BX
= 0  :    PUT     0   IN BX
> 0  :    PUT     1   IN BX
END_CASE
```

في لغة التجميع :

```
CMP AX, 0          ; AX افحص
JL    NEGATIVE   ; AX < 0
JE    ZERO            ; AX = 0
JG    POSITIVE     ; AX > 0
; Otherwise (Else) part will be here
NEGATIVE :
MOV    BX,-1
JMP      END_CASE
```

```
ZERO :
MOV BX,0
JMP END_CASE
POSITIVE :
MOV BX,1
END_CASE:
```

لاحظ أننا نحتاج فقط لــ CMP واحدة لأن أوامر التفرع لاتؤثر علي البيارق.

مثال : إذا كانت محتويات المسجل AL هي الرقم 1 أو الرقم 3 أطبع "O"، وإذا كانت محتويات المسجل AL هي الرقم 2 أو الرقم 4 أطبع 'E'.

الحل :

```
CASE AL of
1,3:DISPLAY "O"
2,4:DISPLAY "E"
END_CASE
```

بلغة التجميع

```
CMP AL , 1
JE ODD
CMP AL , 3
JE ODD
CMP AL , 2
JE EVEN
CMP AL , 4
```

```
JE    EVEN
JMP       END_CASE
ODD:          MOV     DL , 'O'
JMP       DISPLAY
EVEN:         MOV     DL , 'E'
DISPLAY:    MOV     AH ,2
INT 21H
END_CASE:
```

التفرع بشروط مركبة Compound Conditions

في بعض الأحيان يتم استعمال شرط مركب لعملية التفرع مثل

IF   condition1   AND   condition2

IF   condition1   OR condition2

أو

حيث في الحالة الأولي تم استخدام الشرط "و" AND وفي الحالة الثانية تم استخدام الشرط "أو" OR

الشرط "و" AND Condition

تكون نتيجة الشرط "و" صحيحة إذا تحقق كل من الشرطين في آن واحد

مثال: اقرأ حرف من لوحة المفاتيح، وإذا كان حرفاً كبيراً Capital Letter اطبعه

خوارزمية الحل:

Read a Character into AL

```
If  ( 'A' <= character  AND character <= 'Z')
then
Display character
End_IF
```

بلغة التجميع

```
MOV     AH , 1    ; قراءة الحرف
INT 21h
CMP     AL , 'A'
JNGE    End_IF
CMP     AL , 'Z'
JNLE    End_IF
MOV     DL , AL
MOV     AH , 2
INT 21h
End_IF:
```

OR Condition الشرط "أو"

يتحقق الشرط "أو" إذا تحقق أي من الشرطين أو كلاهما

مثال : اقرأ حرف وإذا كان الحرف 'y' أو 'Y' اطبعه وإذا لم يساوي 'y' أو 'Y' قم بإنهاء البرنامج

خوارزمية الحل

```
Read character from keyboard into AL
IF ( character = 'y'      OR  character = 'Y' )
then
Display character
```

Else
Terminate the program
End_IF

بلغة التجميع

```
MOV    AH , 1 ; قراءة الحرف
INT 21h
CMP    AL , 'y'
JE   then
CMP    AL , 'Y'
JE   Then
JMP    else_
Then:   MOV    DL,AL
        MOV    AH , 2
        INT 21h
        JMP    End_if
else:   MOV    AH ,4ch
        INT 21h
End_if:
```

التكرار

التكرار هو عملية تنفيذ مجموعة من الأوامر لأكثر من مرة.وقد يكون التكرار لعدد محدد من المرات أو قد يكون التكرار حتى حدوث حدث محدد.

التكرار لعدد محدد

في هذه الحالة يتم تكرار مجموعة من الأوامر لعدد محدد من المرات وتسمي بالfor loop والشكل العام هو

For loop_count times do
statements
End_for

يتم استخدام الأمر loop لتمثيل الحلقة وهو بالصيغة

loop destination_label

حيث يتم استخدام المسجل CX كعداد ويتم تحميله بقيمة العداد (عدد مرات تكرار الحلقة) وتنفيذ الأمرloop يؤدي إلى إنقاص قيمة المسجل CX بمقدار واحد وإذا لم تصبح قيمة المسجل CX =صفر يتم التفرع إلي العنوان destination_label(الذي يجب أن يسبق العنوان الحالي بمقدار 126 خانة كحد أقصي) ويتم تكرار هذه العملية حتى تصل قيمة المسجل CX إلي الصفر عندها يتم الانتهاء من الحلقة ومواصلة البرنامج. باستخدام loop يكون على النحو التالي

```
وضع قيمة ابتدائية في المسجل( CX ) ;
top:
جسم البرنامج ;
loop top
```

مثال :- اكتب برنامج يستخدم حلقة التكرار وذلك لطباعة 80 نجمة"*"

الحل

```
for 80 times do
    display "*"
End_for
```

بلغة التجميع

```
MOV    CX , 80  ; عدد مرات النجوم المطلوب عرضها
MOV    AH , 2
MOV    DL , '*'
Top:   INT 21h
       LOOP   top
```

من البرنامج السابق نلاحظ أن عملية التكرار باستخدام الأمر LOOP يؤدي إلي تكرار جسم الحلقة مره واحدة علي الأقل وبالتالي إذا كانت قيمة العداد CX تساوي صفر فإن البرنامج سيؤدي جسم الحلقة مرة واحدة حيث

يقوم بطرح 1 من العداد لتصبح قيمة العداد 65535 حيث تقوم الحلقة بالتكرار عدد 65535 (00FFFh) مرة بعدها ينتهي البرنامج.

لعلاج هذه الحالة يجب التأكد من أن قيمة المسجل CX لا تساوي صفر قبل الدخول للحلقة وذلك باستخدام الأمر JCXZ

(Jump if CX is Zero) ويكون شكل البرنامج علي النحو التالي

```
JCXZ        skip
Top:
; جسم الحلقة
looptop
skip:
```

حلقة WHILE

يتم تكرار هذه الحلقة حتى حدوث شرط محدد حيث الشكل العام لها علي النحو التالي

```
While Condition DO
Statements
End_while
```

يتم اختبار الشرط في بداية الحلقة فإذا تحقق الشرط يتم تنفيذ جسم الحلقة وإذا لم يتحقق يتم الخروج من الحلقة وتنفيذ الأوامر التالية في البرنامج.

لاحظ أن الشرط قد لا يتحقق من البداية وبالتالي لا يتم الدخول أصلاً في جسم الحلقة مما يؤدي إلي إمكانية عدم تنفيذ جسم الحلقة علي الإطلاق. لاحظ أيضاً أن جسم الحلقة يقوم دائماً بتغيير أحد معاملات شرط الحلقة حتى يتحقق شرط

إنهاء الحلقة (في حالة عدم تغيير معاملات الشرط تكون الحلقة لانهائية)

مثال :اكتب جزء من برنامج يقوم بإيجاد عدد الحروف في سطر محدد

الحل

INITIALIZE COUNT TO 0 ; ابدأ العداد بالقيمة صفر

READ A CHARACTER ; اقرأ حرف

WHILE CHARACTER<>CARRIAGE-RETURN DO

COUNT =COUNT+1

READ A CHARACTER

END-WHILE

بلغة التجميع:

MOV DX,0 ; عداد الحروف

MOV AH , 1 ; الخدمه رقم 1 (قراءة حرف)

INT 21h

WHILE:

CMP AL,0DH ; من نهاية السطر

JE END_WHILE ; اذا كانت نهاية السطر

INC DX ; أضف واحد إلى ا
لعداد
INT 21H ; اقرأ الحرف التالي
JMP WHILE.
END-WHILE :

حلقة REPEAT

وهى حلقة أخري تقوم بالتكرار حتى حدوث شرط محدد.والشكل العام لها يكون على الصورة

```
REPEAT
STATEMENT(s) ;
UNTIL CONDITION
```

وهنا يتم تنفيذ جسم الحلقة ثم بعد ذلك يتم اختبار الشرط .فإذا تحقق الشرط يتم الخروج من الحلقة أما إذا لم يتحقق يتم تكرار الحلقة .

مثال :اكتب جزء من برنامج يقوم بقراءة حروف تنتهي بالمسافة blank

```
MOV AH,1 ; خدمة قراءة حرف
REPEAT:
INT 2!H
CMP AL ,' '; قارن الحرف والمسافة
JNE REPAET ; اذا لم يساويه كرر الحلقه
```

الفرق بين حلقة WHILE وحلقة REPEAT

استخدام الحلقتين عادة يعتمد على تفضيل الشخص وعموما تمتاز حلقة WHILE بان الشرط يتم اختباره قبل الدخول إلى الحلقة وبالتالي يمكن عدم تنفيذ جسم الحلقة على الإطلاق بينما تمتاز حلقة REPEAT بالمرور على جسم الحلقة أولاً ثم اختبار الشرط وبالتالي يجب تنفيذ جسم الحلقة مرة واحدة على الأقل.

كتابة برنامج

لتوضيح كيفية كتابة برامج كبيره من لغة راقية إلي لغة التجميع نوضح المثال التالي :

اكتب برنامج كامل يقوم بسؤال المستخدم لإدخال جمله يقوم البرنامج بتحديد أصغر حرف كبير ورد في الرسالة وأكبر حرف كبير يرد فى الرسالة (وذلك حسب ترتيب الحروف في جدول الـ ASCII).

إذا لم ترد حروف كبيره يقوم البرنامج بإظهار الرسالة (No capital letters) . كالآتي :

TYPE A LINE OF TEXT :
SUDAN UNIVERSITEY OF SCIENCE
AND TECHNOLOGY

FIRST CAPITAL = A LAST CAPITAL = Y

سوف نقوم بكتابة هذا البرنامج على طريقة تجزئه المشكلة إلي مجموعه من المشاكل الفرعية الصغيرة التي يتم حل كل واحدة منها على حده وهذه الطريقة تسمى بطريقه التصميم من أعلى إلي اسفل TOP - DOWN PROGRAM DESIGN كالآتي :

1- اظهر رسالة للمستخدم لإدخال نص.

2- اقرأ وتعامل مع النص .

3- اظهر النتيجة .

وبعد ذلك يتم التعامل مع كل خطوه بالتفصيل.

1- إظهار الرسالة للمستخدم لإدخال نص

يتم ذلك عن طريق كتابة الجزء التالي

```
MOV     AH,9          ;  خدمة رقم 9 نص
LEA     DX ,PROMPT ; عنوان الرسالة
INT     21H           ; اعرضها
```

حيث يتم تعريف الرسالة PROMPT في مقطع البيانات على النحو التالي

```
PROMPT DB  ' TYPE   A LINE OF TEXT :
',0DH,0AH, '$'
```

وهى تتضمن تحويل المحث CURSOR إلى السطر التالي

2- قراءة النص والتعامل معه:

هذه الخطوة تحتوي على قلب البرنامج والتي يتم فيها الجزء الكبير في البرنامج ويمكن كتابة الخوارزمية لها على النحو التالي

```
Read Character; اقرا حرف
While Character Is Not a Carriage Return Do
    IF Character Is A Capital Letter Then
        IF Character Precedes First Capital THEN
            First Capital =CHARACTER
        END_IF
        IF Character Follows Last Capital THEN
            Last Capital = Character
        END_IF
    END_IF
    Read Character
END_WHILE
```

حيث يكون الحرف كبير إذا تحقق الشرط Character >= 'A' AND Character <='Z'

ويكون هذا الجزء بلغة التجميع علي النحو التالي

```
MOV AH , 1
INT 21H
WHILE:
CMP AL,0DH
```

```
JE   END_WHILE
CMP      AL , 'A'
JNGE    END_IF
CMP      AL , 'Z'
JNLE     END_IF
CMP      AL,FIRST
JNL       CHECK-LAST
MOV      FIRST,AL
CHECK-LAST:
CMP      AL,LAST
JNG       END-IF
MOV      LAST,AL
END_IF:       INT  21H
JMP WHILE
END_WHILE  :
```

حيث FIRST وLAST عبارة عن متغيرات حرفية يتم تعريفها

في مقطع البيانات على النحو التالى:-

```
FIRST    DB       ']'
LAST          DB       '@'
```

حيث الحرف] هو الحرف التالي للحرف Z و الحرف @ هو

الحرف السابق للحرف A

3/ طباعة النتيجة :-

فى هذه الخطوة يتم التالي :

IF NO CAPITAL LETTER TYPED THEN

```
DISPLAY  'NO CAPITAL'
ELSE
DISPLAY FIRST & LAST
CHARACTER
END_IF
```

حيث يتم إظهار الرسالة الأولى في حالة عدم إدخال أي حرف كبير داخل الرسالة أو قيمة اكبر واصغر حرف تم إدخاله. ولأجراء ذلك نقوم بتعريف البيانات التالية:

```
NOCAP-MSG     DB      'NO CAPITALS $'
CAP-MSG    DB       'FIRST CAPITAL='
FIRST         DB       ']'
DB  'LAST CAPITAL='
LAST    DB  '@   $'
```

و يتم كتابة الجزء التالي

```
MOV  AH ,  9
CMP  FIRST,']'
JNE  CAPS
LEA  DX ,NOCAP_MSG
JMP      DISPLAY
CAPS   :       LEA       DX, CAP_MSG
DISPLAY:     INT 21H
```

البرنامج الكامل

```
TITLE THIRD: CASE CONVERSION PROGRAM
.MODEL SMALL
```

```
.STACK 100H
.DATA
CR          EQU      0DH
LF          EQU      0AH
PROMPT       DB  'TYPE A LINE OF
TEXT',CR,LF,'$'
NOCAP_MSG       DB  CR,LF,'NO
CAPITALS $'
CAP_MSG    DB  CR,LF,'FIRST CAPITAL
='
FIRST    DB  ']'
DB  ' LAST CAPITAL = '
LAST             DB  '@  $'
.CODE
MAIN PROC
; initialize DS
MOV       AX,@DATA
MOV       DS,AX
;display opening message
LEA DX,prompt
MOV       AH,09H
INT 21H
;read and process a line of text
MOV       AH,01H
INT 21H
WHILE_:
CMP       AL,CR
```

```
JE   END_WHILE
;if char is capital
CMP      AL,'A'
JNGE     END_IF
CMP      AL,'Z'
JNLE     END_IF
; if character precede first capital
CMP      AL,FIRST
JNL CHECK_LAST
MOV      FIRST,AL
CHECK_LAST:
; if character follow last capital
CMP      AL,LAST
JNG      END_IF
MOV      LAST,AL
END_IF:
INT 21H
JMP      WHILE_
END_WHILE:
MOV      AH,9
;if no capital were typed
CMP      FIRST,']'
JNECAPS
LEA      DX,NOCAP_MSG
JMP      DISPLAY
CAPS:
LEADX,CAP_MSG
```

```
DISPLAY:
INT 21H
;exit to DOS
MOV     AH,4CH
INT 21H
MAIN    ENDP
END     MAIN
```

تمارين

1 – حول العبارات التالية إلي لغة التجميع

```
1 -   IF    AX < 0    THEN
PUT   -1  IN  BX
END_IF
2 -   IF    AL < 0          THEN
PUT  FFh      IN   AH
ELSE
PUT  0    IN   AH
END_IF
3 -   IF  (  DL  >= "A"     AND   DL  = <  "Z" )
Then
DISPLAY      DL
END_IF
4 -   IF    AX  <  BX  THEN
IF    BX  <  CX  THEN
PUT        0  IN  AX
ELSE
PUT 0    IN BX
END_IF
END_IF
5 -   IF  (  AX  <  BX  )  OR  (  BX  <  CX  )
THEN
PUT    0    IN  DX
ELSE
PUT    1    IN DX
```

```
END_IF
```

```
6 -   IF  AX <  BX  THEN
          PUT  0  IN  AX
                    ELSE
      IF   BX  <  CX    THEN
          PUT  0  IN  BX
                    ELSE
          PUT  0  IN CX
                  END_IF
                  END_IF
```

2 – استعمل الشكل الهيكلي لعبارة CASE اكتب الجزء التالي من البرنامج بلغة التجميع

أ – اقرأ حرف.

ب – إذا كان الحرف 'A' اطبع (نفذ) Carriage Return

جـ – إذا كان الحرف 'B' اطبع (نفذ) Line Feed

د – إذا كان أي حرف آخر قم بإنهاء البرنامج والعودة لنظام التشغيل.

3 – اكتب جزء من برنامج يقوم بالآتي :

أ – ضع حساب مجموع الأرقام 1 + 4 + 7 + ... +
148 في المسجل AX.

ب – ضع حساب مجموع الأعداد 100 + 95 + 90
+ ...... + 5 في المسجل BX.

4 – مستخدماً الأمر LOOP قم بكتابة برنامج يقوم بالآتي :

أ – حساب أول 50 عنصر في المتوالية 1 ، 5 ، 9 ،
13 في المسجل AX

ب – قراءة حرف وطباعته 80 مرة في السطر التالي.

5 – الخوارزمية التالية تقوم بقسمة رقمين باستخدام عملية الطرح

```
INITIALIZE QUOTIENT TO 0
WHILE DIVIDENT  > =  DIVISOR DO
INCREMENT QUOTIENT
SUBTRACT DIVISOR FROM
DIVIDEND
END_WHILE
```

اكتب جزء من برنامج يقوم بقسمة الرقم الموجود في المسجل AX علي الرقم الموجود
بالمسجل BX ووضع النتيجة في المسجل CX

6 – الخوارزمية التالية تقوم بإيجاد حاصل ضرب رقمينN و M باستخدام عملية الجمع المتكرر

```
INITIALIZE PRODUCT TO 0
```

```
REPEAT
ADD M TO PRODUCT
DECREMENT N
UNTIL  N = 0
```

اكتب جزء من برنامج يقوم بضرب الرقم الموجود في المسجل AX في الرقم الموجود

بالمسجل BX ووضع النتيجة في المسجل CX (يمكنك تجاهل حدوث عملية الفيضان)

7 - إذا علمت أن الأمرين LOOP و LOOPZ يتضمن تنفيذهما إنقاص قيمة المسجل CX وإذا

كانت CX <> 0 و (AND) ZF = 1 يتم تكرار الحلقة (يتم القفز إلي العنوان المحدد).

كذلك الأمرين LOOPNE و LOOPNZ يتضمن تنفيذهما إنقاص قيمة المسجل CX وإذا

كانت CX <> 0 و (AND) ZF = 0 يتم تكرار الحلقة (يتم القفز إلي العنوان المحدد).

اكتب برنامج يقرأ حروف تنتهي إما بالضغط علي مفتاح الإدخال Carriage Return او يتم

إدخال 80 حرف (استعمل الأمر LOOPNE).

<u>البرامج</u>

8 - اكتب برنامج يقوم بإظهار الحرف '?' ثم يقوم بقراءة حرفين كبيرين. يقوم البرنامج بطباعة

الحرفين بعد ترتيبهما في السطر التالي.

9- اكتب برنامج يقوم بطباعة الحروف ابتداء من الحرف رقم 80h وحتي الحرف الرقم FFh من حروف الـ

ASCII يقوم البرنامج بطباعة 10 حروف في السطر

الواحد تفصلها مسافات.

10-اكتب برنامج يقوم بسؤال المستخدم لإدخال رقم سداسي عشر مكون من خانة واحدة (

"0" إلى "9" أو "A" إلى "F") يقوم البرنامج بطباعة القيمة المناظرة في النظام العشري

في السطر التالي. يقوم البرنامج بسؤال المستخدم إذا كان يريد المحاولة مرة ثانية فإذا ضغط

علي الحرف 'Y' أو الحرف 'y' يقوم البرنامج بتكرار العملية وإذا أدخل أي حرف آخر يتم

إنهاء البرنامج. (إذا ادخل المستخدم أي رقم غير مسموح به يقوم البرنامج بإظهار رسالة

والمحاولة مرة أخرى)

11-كرر البرامج في 10 بحيث إذا فشل المستخدم في إدخال رقم سداسي عشر في عدد 3 محاولات

يقوم البرنامج بالانتهاء والعودة إلي نظام التشغيل.

الفصل السادس

الأوامر المنطقية وأوامر الإزاحة والدوران

الأوامر المنطقية AND,OR,XOR

تستخدم الأوامر المنطقية في التعامل مع خانة ثنائية واحدة في المسجل المحدد والشكل العام للأوامر هو:

AND DESTINATION , SOURCE

OR DESTINATION , SOURCE

XOR DESTINATION , SOURCE

وتم تخزين النتيجة في المستودع DESTINATION الذي يجب أن يكون مسجل أو موقع في الذاكرة بينما المعامل الآخر SOURCE يمكن أن يكون مسجل أو موقع في الذاكرة أو قيمة ثابتة. عموماً لا يمكن التعامل مع موقعين في الذاكرة.

يكون تأثر البيارق على النحو التالي :

PF,ZF,ZF : تعكس حالة النتيجة.

AF : غير معرفة.

CF,OF : تساوي صفر .

أحد الاستخدامات المهمة للأوامر المنطقية هو تغيير خانة محددة داخل مسجل ويتم ذلك باستخدام حجاب MASK حيث يتم بواسطته تحديد الخانة المطلوب التعامل معها ويتم الاستعانة بالخصائص التالية للأوامر المنطقية :

b AND 1 = b , b AND 0 = 0
b OR 1 = 1 , b OR 0 = b
b XOR 1 =~ b , b XOR 0 = b

وعلى هذا يمكن الآتي :

1- لوضع القيمة '0' في خانة (أو خانات) محددة Clear يتم استخدام الأمر AND حيث يتم وضع القيمة '0' في الحجاب MASK للخانات المطلوب وضع '0' فيها بينما يتم وضع القيمة '1' في الخانات الغير مطلوب تعديلها .

2- لوضع القيمة '1' في خانة (أو خانات) محددة SET يتم استخدام الأمر OR حيث يتم وضع القيمة '1' في الحجاب MASK للخانات المطلوب وضع '1' فيها بينما يتم وضع القيمة '0' في الخانات الغير مطلوب تعديلها.

3- لعكس قيمة خانة (أو خانات) محددة COMPLEMENT يتم استخدام الأمر XOR حيث يتم

وضع القيمة '1' في الحجاب MASK للخانات المطلوب عكس قيمتها بينما يتم وضع القيمة '0' في الخانات الغير مطلوب تعديلها .

مثال:

ضع القيمة '0' في خانة الإشارة في المسجل AL واترك باقي الخانات بدون تعديل.

الحل

يتم استخدام القيمة 0111 1111b = 7Fh كحجاب MASK ويتم استخدام الأمر AND

AND AL, 7Fh

مثال

ضع القيمة '1' Set في كل من الخانة ذات الوزن الأكبر MSB والخانة ذات الوزن الأصغر LSB في المسجل AL وأترك باقي الخانات بدون تعديل

الحل

يتم استعمال الحجاب Mask = 1000 0001b = 81h ونستخدم الأمر OR كالتالي

OR AL , 81h

مثال

غير إشارة المسجل DX

الحل

يتم استخدام العجاب Mask التالي 1000 0000 0000 0000b = 8000h ونستخدم الأمر XOR

XOR DX , 8000h

وعموماً يتم استخدام الأوامر المنطقية في مجموعة من التطبيقات والتي سنتحدث عن بعضها في الجزء التالي

تحويل الحروف الصغيرة لحروف كبيرة

نعلم أن الحروف الصغيرة ('a' to 'z') تقع في جدول الـ ASCII ابتداء من الرقم 61h وحتى 7Ah بينما تقع الحروف الكبيرة ('A' to 'Z') في جدول الـ ASCII ابتداء من الرقم 41h وحتى 5Ah وعلي ذلك فإنه لتحويل الحرف من صغير إلي كبير نطرح الرقم 20h فمثلاً إذا كان المسجل DL يحتوي علي حرف صغير ومطلوب تحويله إلي حرف كبير نستعمل الأمر SUB DL , 20h وقد قمنا باستخدام هذه الطريقة من قبل. ونريد هنا استخدام طريقة أخري للتحويل.

إذا نظرنا للأرقام المناظرة للحروف نجد أن

الرقم المناظر للحرف 'a' هو 61h = 01<u>1</u>0 0001

الرقم المناظر للحرف 'A' هو 61h = 01<u>0</u>0 0001

ومن الأرقام نلاحظ تحويل الحرف من صغير إلي كبير يتطلب وضع القيمة '0' في الخانة السادسة في المسجل الذي يحوي

الحرف ويتم ذلك باستخدام الحجاب Mask التالي 1101 1111b= 0DFh ونستعمل الأمر AND

AND DL , 0DFh

ويمكنك الآن توضيح كيفية تحويل الحروف الكبيرة إلي حروف صغيرة بنفسك.

تفريغ مسجل (وضع صفر فيه) Clear Register

نعلم أنه لوضع القيمة صفر في مسجل يمكننا استخدام أحد الأمرين MOV AX,0

أو SUB AX , AX إذا أردنا استخدام أمر منطقي يمكننا الاستعانة بالأمر XOR حيث نعلم أن

0 XOR 0 = 0 و 1 XOR 1 = 0

وبالتالي يمكننا استخدام الأمر XOR للمسجل مع نفسه لنضع الرقم صفر فيه علي النحو التالي

XOR AX , AX

اختبار وجود الرقم صفر في مسجل

لأن '0' OR '0' = '0' و '1' OR '1' = '1' فإن الأمر OR AX , AX يبدو كأنه لا يفعل شيئاً حيث لا يتم تغيير محتويات المسجل AX بعد تنفيذ الأمر، ولكن الأمر يقوم بالتأثير علي بيرق الصفر ZF و بيرق الإشارة SF فإذا كان المسجل AX يحوي الرقم صفر فسيتم رفع بيرق الصفر (ZF =

(1 وبالتالي يمكن استخدام هذا الأمر بدلاً من استخدام الأمر
CMP AX , 0

الأمر NOT

يقوم الأمر NOT بحساب المكمل لواحد 1's Complement (وهو تحويل الـ '0' إلـي '1' والـــ '1' إلـي '0' أي عكـس الخانات بداخل المسجل) والشكل العام للأمر هو :

NOT Destination

ومثال له الأمر NOT AX

الأمر TEST

يقوم الأمر TEST بعمل الأمر AND ولكـن بـدون تغييـر محتويات المستودع Destination والهدف منه هـو التـأثير علي بيارق الحالة والشكل العام للأمر هو

TEST Destination , Source

ويقوم بالتأثير علي البيارق التالية:

البيارق PF و ZF و SF تعكس النتيجة

البيرق AF غير معرف

البيارق OF و CF تحتوي علي الرقم 0

إختبار خانة أو خانات محددة

يستخدم الأمر TEST لاختبار محتويات خانة أو خانات محددة ومعرفة إن كان بها '1' أو '0' حيث يـتم اسـتخدام حجـاب Mask ووضع الرقم '1' في الخانات المطلوب اختبارها ووضع

الرقم '0' في الخانات الغير مطلوب معرفة قيمتها وذلـك لأن 1 AND b = b و 0 AND b = 0 ويـتم استخدام الأمر

TEST Destination , Mask

وبالتالي فإن النتيجة ستحتوي علي الرقم '1' في الخانة المراد اختبارها فقط إذا كانت هذه الخانة تحتوي علي الـرقم '1'، وتكون صفر في كل الخانات الأخري.

مثال:

اختبر قيمة المسجل AL وإذا احتوى على رقم زوجي قم بالقفز إلى العنوان Even_No

الحل

الأرقام الزوجية تحتوي على الرقم 0 في الخانة ذات الوزن الأصغر LSB وعلى ذلك لاختبار هذه الخانة يتم استخدام الحجاب MASK التـالي 1b0000000 ويكون البرنامج على الصورة التالية :

```
TEST    AL , 01h
JZ      Even_No
```

أوامر الإزاحة:

تستخدم أوامر الإزاحة لإجراء عملية إزاحة بمقدار خانة أو أكثر للخانات الموجودة في المستودع وذلك لليمين أو لليسار .

عند استخدام الأمر shift يتم فقد للخانة التي يتم إزاحتها إلى الخارج ، بينما في حالة أوامر الدوران يتم دخول هذه الخانة إلى الطرف الثاني من المستودع ،كما سنرى فيما بعد.

يوجد شكلان لأوامر الإزاحة وهي إما :

Opcode Destination,1

أو

Opcode Destination,CL

حيث يحتوي المسجل CL على عدد مرات الإزاحة المطلوب تنفيذها .

الإزاحة لليسار (SHL) Shift Left :

يقوم الأمر SHL بعمل إزاحة لليسار ويمكن أن تكون الإزاحة بمقدار خانة واحدة وفي هذه الحالة نستعمل الأمر:

SHL Destination , 1

أو أكثر من خانة حيث يتم وضع عدد مرات الإزاحة المطلوبة في المسجل CL واستعمال الأمر

SHLDestination , CL

ولا تتغير قيمة المسجل CL بعد تنفيذ الأمر

تقوم البيارق PF , SF , ZF بتوضيح حالة النتيجة .

البيرق CF يحتوي على آخر خانة تمت إزاحتها للخارج

بينما البيرق of يحتوي على 1 إذا كانت آخر عملية إزاحة أدت إلى رقم سالب .

مثال:

إذا كان DH = 8AH و CL = 3 ما هي محتويات المسجلين CL و DH بعد تنفيذ الأمر SHL DH , CL وكذلك بيرق المحمول.

الحل:

قبل تنفيذ الأمر كانت محتويات المسجل DH هي الرقم 10001010 بعد 3 ازاحات إلي اليسار تصبح محتوياته = h 50 01010000 بينما يحتوى المسجل CL على قيمته السابقة (الرقم 3) ويحتوى بيرق المحمول على القيمة '0'. (محتويات DH الجديدة يمكن الحصول عليها بمسح 3 أرقام ثنائية في أقصى اليسار وإضافة 3 أصفار في أقصى اليمين)

الضرب باستخدام الإزاحة لليسار:

تعتبر عملية الإزاحة لليسار عملية ضرب في الرقم (2d) مثلاً الرقم 101 (5d) إذا تمت إزاحته لليسار بمقدار خانه واحدة نحصل على الرقم 1010 (10d) وبالتالي فإذا تمت الإزاحة بمقدار خانتين تعتبر كأننا قمنا بضرب الرقم في العدد (4d) وهكذا. وبالتالي فإن الإزاحة لليسار في رقم ثنائي تعني ضربه في (2)

الأمر Shift Arithmetic Left (SAL):

يعتبر الأمر SAL مثل الأمر SHL ولكن يستخدم SAL في العمليات الحسابية حيث يقوم الأمرين بتوليد نفس لغة الآلة Machine Code.

الفيضان:

بالرغم من أن عملية ألإزاحه تقوم بالتأثير على بيارق الفيضان والمحمول إلا انه إذا حدثت ازاحه لأكثر من مره فان حالة البيارق لا تدل على أي شئ حيث أن المعالج يعكس فقط نتيجة أخر عملية ازاحه فمثلاً إذا حدثت عملية ازاحه لمسجل يحتوى على الرقم 80h وذلك بمقدار خانتين CL=2 فسنجد أن قيمة البيارق Of , Cf تساوى صفر وذلك بالرغم من حدوث عملية الفيضان.

مثال: أكتب الأوامر اللازمة لضرب محتويات المسجل AX في الرقم (8) مفترضاً عدم وجود فيضان.

الحل: نحتاج إلي إزاحة لليسار بمقدار (3) خانات.

```
MOV  CL , 3
SAL AX, CL
```

الازاحة لليمين والأمر Shift Right (SHR):

؟ يقوم الأمر SHR بعمل ازاحه لليمين للمستودع ويأخذ الصورة SHR Destination,1 يتم إدخال القيمة صفر في الخانة ذات الوزن الأعلى MSB بينما يتم إزاحة الخانة ذات الوزن الأصغر LSB إلى بيرق المحول Cf . كبقية أوامر الازاحه يمكن إجراء عملية الازاحه لأكثر من خانه وذلك بوضع عدد مرات الازاحه المطلوبة في المسجل CL واستخدام الصيغة.

```
SHR Destination, CL
```

ويكون تأثر البيارق كما في حالة الأمر SHL.

مثال:

ما هي محتويات المسجل DH و والبيرق CF بعد تنفيذ الجزء التالي من برنامج

```
MOV     DH, 8Ah
MOV     CL , 2
SHR     DH,CL
```

الحل:

DH = 10001010

بعد الازاحه بمقدار خانتين تصبح محتويات المسجل

DH = 00100010 = 22h

وتكون قيمة البيرق Cf هى '1'

الامر Shift Arithmetic Right (SAR):

يقوم الأمر SAR بنفس عمل الأمر SHR ماعدا أن محتويات الخانة ذات الوزن الأعلى MSB لا يتم تغييرها بعد تنفيذ الأمر. وكبقية أوامر الازاحه يأخذ الأمر الصيغة.

SAR Destination , 1

أو فى حالة الازاحه عدد من المرات حيث يتم وضع عدد مرات الإزاحة المطلوب في المسجل CL ويأخذ الأمر الصيغة

SAR Destination, CL

القسمة باستخدام الازاحه لليمين:

يتم استخدام الازاحه لليمين لإجراء عملية القسمة على العدد 2 وذلك في حالة الأعداد الزوجية. أما بالنسبة للأعداد الفردية فان النتيجة تكون مقربه للعدد الصحيح الأصغر وتكون قيمة بيرق المحمول Cf تساوى 1 فمثلاً عند إجراء عملية الازاحه

لليمين للرقم 5=(00000101) فان النتيجة هي الـــرقم (00000010) وهو الرقم 2.

القسمة بإشارة وبدون إشارة:

عند إجراء عملية القسمة يجب التفرقة بين الأرقام بإشارة والأرقام بدون إشارة. فـــي حالـــة الأرقـــام بدون إشارة يمكن استخدام الأمـــر SHR . بينمـــا في حالة الأرقام بإشارة يجب استخدام الأمر SAR حيث يتم الاحتفاظ بإشارة الرقم (تذكر أن خانـــة الإشارة هي الخانة ذات الوزن الأكبر).

مثال:

استخدم الازاحه لليمين لقسمة الرقم 65143 على الـــرقم 4 وضع النتيجة في المسجل AX.

الحل:

```
MOV     AX, 65143
MOV     CL,2
SHR     AX, CL
```

مثال:

إذا احتوى المسجل AL على الرقم 15- ما هي محتويات المسجل AL بعد تنفيذ الأمر.

```
SAR     AL,1
```

الحل:

تنفيذ الأمر يعنـى قسـمة محتويـات المسـجل AL بالعدد 2 ويتم تقريب النتيجة كما ذكرنـا وهنـا النتيجة هي الـرقم 7.5- وبتقريبـه الـى العـدد الأصغر ونحصل على العدد 8- وإذا نظرنا للعـدد في الصـورة الثانيـة نجـد أن العـدد 15- هـو 11110001 وبعد إجراء عملية الازاحه لليمين نحصل على الرقم 11111000 وهو العدد 8-.

عموماً يمكن استخدام أوامر الازاحه لليسار ولليمين لإجراء عمليتي الضرب والقسمة علـى العـدد 2 أو مضاعفاته وإذا أردنا إجراء عملية الضـرب علـى إعداد غير العدد 2 ومضاعفاته يتم إجراء عمليـة إزاحة وجمع كما سنرى فيمـا بعـد كمـا يمكـن استخدام الأوامـر IMUL, MUL للضـرب والأوامـر IDIV, DIV لإجراء عمليـة القسـمة علـى أي رقـم ولكن تعتبر هذه الأوامر أبطأ من عملية الازاحه.

أوامر الدوران:

الدوران لليسار (ROL) Rotate Left

يقوم هذا الأمر بإجراء عملية ازاحه لليسـار ويـتم وضع الخانة ذات الوزن الأعلى فـي الخانـة ذات الوزن الأصغر وفى نفس الوقت يتم وضـعها فـي

بيرق المحمول CF . ويتم النظر للمسجل كأنه حلقة كاملة حيث الخانة ذات الوزن الأعلى بجوار الخانة ذات الوزن الأصغر ويأخذ الأمر الصور

```
ROL     Destination , 1
ROL     Destination , CL
```

الدوران لليمين: Rotate Right (ROR)

يقوم هذا الأمر بنفس عمل الأمر ROL فيما عدا أن الازاحه تكون لليمين حيث يتم وضع الخانة ذات الوزن الأصغر في الخانة ذات الوزن الأكبر وفى نفس الوقت يتم وضعها في بيرق المحمول. ويأخذ الأمر أحد الصيغتين:

```
ROR     Destination ,1
ROR     Destination ,CL
```

يلاحظ انه في الأمرين ROR , ROL يتم وضع الخانة التي يتم طردها في بيرق المحمول CF

مثال:

استخدم الأمر ROL لحساب عدد الخانات التي تحتوى على الرقم (1) في المسجل BX دون تغيير محتويات المسجل BX. ضع النتيجة في المسجل Ax.

الحل:

MOV DX ,16D ; عــدد التكــرار
للالتفاف

XOR AX,AX ; AX يتم حساب
عدد الخانات في

MOV CX,1 ;عدد الخانات

Top : ROL BX,CX ; CF الخانة
التي تم طردها توجد في

JNCNEXT
INC AX
NEXT: DEC DX
JNZ Top

(RCL) Rotate الدوران لليسار عبــر بيــرق المحمــول through Carry Left

يقوم هذا الأمر بـإجراء عمليــة الــدوران لليســار واعتبار بيرق المحمول جزء من المسجل حيث يــتم وضع الخانة ذات الوزن الأعلى في بيرق المحمول ويتم وضع محتويات بيرق المحمول في الخانة ذات الوزن الأصغر. ويأخذ إحدى الصيغتين.

RCL Destination , 1
RCL Destination , CL

Rotate through الدوران لليمين عبر بيــرق المحمــول carry Right RCR

يقوم هنا الأمر بنفس عمل الأمر RCL فيما عدا أن الدوران يكون لليمين حيث يتم وضع الخانة ذات الوزن الأصغر في بيرق المحول ووضع بيرق المحول في الخانة ذات الوزن الأعلى ويأخذ الصيغتين

RCR Destination , 1
RCR Destination ,CL

مثال:

إذا كانت محتويات المسجل DH هي الرقم 8Ah وكانت محتويات بيرق المحول هي الرقم CF=1 والمسجل CL يحتوى على الرقم 3 ما هي محتويات المسجل DH وبيرق المحمول بعد تنفيذ الأمر

RCR DH, CL

الحل:

	DH	CF
القيمة الابتدائية	10001010	1
بعد الدوره الاولى نحو اليمين	11000101	0
بعد الدوره الثانية نحو اليمين	01100010	1
بعد الدوره الثالثة	10110001	0

نحو اليمين

أي محتويات المسجل DH هي الرقم B1h وبيرق المحمول يساوى صفر.

مثال:

أكتب جزء من برنامج يقوم بعكس الخانات الموجودة في المسجل AL ووضع النتيجة في المسجل DL فمثلاً إذا كانت محتويات المسجل AL هي الرقم الثاني 11011100 يتم وضع الرقم 00111011 في المسجل BL.

الحل:

يتم استخدام الأمر SHL حيث يتم وضع الخانة ذات الوزن الأكبر في بيرق المحول وبعدها مباشرة يتم استخدام الأمر RCR لوضعها في الخانة ذات الوزن الأعلى في المسجل BL وتكرار هذه العملية عدد 8 مرات. كما في الجزء التالي

```
          MOV     CX, 8
Reverse:          SHLAL,1
          RCR     BL,1
          Loop    Reverse
          MOV     AL , BL
```

قراءة وطباعة الأرقام الثنائية والسداسية عشر:

في هذا الجزء سنتناول كيفية كتابة برامج تقــوم بقراءة أرقام ثنائية أو سداسية عشــر مــن لوحــة المفاتيح وكذلك طباعة الأرقام في الصورة الثنائية والسداسية عشر في الشاشة.

1- إدخال الأرقام الثنائية:

في برنامج الإدخال للأرقام الثنائية يقوم المســتخدم بإدخال رقم ثنائي انتهــى بالضــغط علــى مفتــاح الإدخــال Carriage Return. حيــث يكــون الــرقم المدخل عبارة عن سلسة الحــروف '0' و '1' وعنــد إدخال كل حرف يتم تحويله إلى القيمــة النــاظرة (1, 0) ونجمع هذه الخانات في مسجل. الخوارزميــة التالية تقوم بإدخال رقم ثنائي مــن لوحــة المفــاتيح ووضعه في المسجل BX :

```
Clear  BX ( BX will hold Binary values )
Input a character (  ‘0’  OR ‘1’)
While  character < > CR DO
      Convert character to binary value
      Left shift BX
      Insert value into LSB of BX
      Input a character
End_While
```

ويمكن توضيح الخوارزمية في حالة إدخال الرقم 110 كالتالي:

Clear BX : BX = 0000 0000 0000 0000

Input character '1' , convert to 1

Left shift BX: BX = 0000 0000 0000 0000

Insert value into LSB of BX: BX = 0000 0000 0000 0001

Input character '1', convert to 1

Left shift BX: BX = 0000 0000 0000 0010

Insert value into LSB of BX: BX = 0000 0000 0000 0011

Input character '0' , convert to 0

left shift BX : BX = 0000 0000 0000 0110

Insert value into LSB of BX

BX = 0000 0000 0000 0110

محتويات المسجل BX هي 110b

تفترض الخوارزمية السابقة أن الأرقام المدخلة تحتوى على '0' و '1' فقط وأن عدد الخانات لا يتعدى 16 خانة وإلا سيتم فقد أول خانه تم إدخالها في حالة إدخال 17 خانة وأول خانتين إذا تم إدخال 18 خانه وهكذا.

تم عمل ازاحه للمسجل BX لليسار لفتح خانة في المسجل BX في الخانة ذات الوزن الأصغر وإدخال الرقم المدخل في الخانة المفتوحة باستخدام الأمر OR حيث أن الخانة ذات الوزن الأصغر تحتوى على الرقم 0 (نتيجة للإزاحة لليسار والتي تضع الرقم 0 فيها) ونعلم أن b OR 0 = b وبالتالي فانه بعد استخدام الأمر OR تصبح القيمة المخزنة فى الخانة ذات الوزن الأصغر هي قيمة الرقم المدخل ويصبح هذا الجزء من البرنامج بلغة التجميع على النحو التالي:

```
    XOR     BX,BX
    MOV     AH,1
    INT 21h        ; اقرأ حرف
While_:
    CMP     AL, 0Dh
    JE   END_While
    AND     AL , 0fh       ;   حول الحرف إلى
رقم ثنائي
    SHL     BX, 1
    OR  BL, AL   ; BL  ادخل القيمة في الخانة
 ذات الوزن الأصغر في
    INT 21h        ; اقرأ الحرف التالي
    JMP   While_
```

```
END_While:
```

2 - إخراج الأرقام الثنائية Binary Output:

في حالة إخراج الرقم في الصورة الثنائية نستخدم عملية الدوران لليسار حيث يتم إزاحة الخانة ذات الوزن الأكبر إلى بيرق المحمول. ويتم اختبار محتويات البيرق فإذا كانت تساوى 1 يتم طباعة الحرف '1' وإذا كانت تساوى صفر يتم طباعة الحرف '0'. وفيما يلي خوارزمية البرنامج

```
FOR 16 times Do
    Rotate left BX
    If   CF = 1 then
        Output '1'
    else
        Output   '0'
    end - if
END_FOR
```

البرنامج بلغة التجميع يُترك كتمرين للطالب .

3 - إدخال الأرقام السداسية عشر Hex input:

الأرقام السداسية عشر المدخلة تحوى المفردات '0' إلى '9' والحروف 'A' إلى 'F' تنتهي بمفتاح الإدخال في نهاية الرقم. وللتبسيط سنفترض هنا أن الحروف المدخلة حروف كبيره فقط وان المدخلات لا تتعدى 4 خانات سداسية عشر (السعه القصوى للمسجل). طريقة

عمل الخوارزمية هي نفسها الطريقة المتبعة في إدخال الأرقام الثنائية فيما عدا أن عملية الازاحه للمسجل تـــتم بأربعـــة إزاحـــات فـــي المـــرة الواحـــدة (لان الخانـــة السداسية عشر يحتوى على أربعة خانات ثنائية) وذلك لتفريغ مكان لإدخال الخانة السداسية عشر فيه. وفيمـــا يلي نذكر خوارزمية البرنامج:

```
Clear        BX
Input  Hex character
While character <> CR Do
        Convert character to Binary value
        Left shift BX 4 Times
        Insert value into lower 4 bits of BX
        input a character
End_While
```

ويكون البرنامج بلغة التجميع كما يلي:

```
XOR      BX , BX
MOV      CL,4
MOV      AH,1
INT 21h              ; اقرأ أول حرف
While_:
CMP      AL , 0dh
JE   END_While
; حول الحرف أى الصورة الثنائية
```

```
CMP     AL , 39h        ; قارن مع الحرف "9"
JG  Letter          ; اذا كان اكبر فهو حرف
;المفردة عبارة عن رقم
AND     AL , 0fh ; حول إلى رقم ثنائي
JMP     shift
;المفردة عبارة عن حرف
Letter:  Sub     AL , 37h      ; حول إلى رقم ثنائي
Shift:   SHL     BX, CL
; BX ادخل القيمة في المسجل
OR      BL, AL ; ضع القيمة في الأربع خانات السفلي
INT 21h ;    اقرأ الحرف الثاني
JMP     While_
END_While:
```

4- إخراج الأرقام السداسية عن HEX Output:

يحتوى المسجل BX على 16 خانة ثنائية أي 4 خانات سداسية عشر. ولطباعة هذا الرقم في

الصورة السداسية عشر نبدأ من اليسار ونأخذ آخر أربعة خانات ثم نحولها إلى خانه سداسية عشر ونطبعها ونستمر كذلك 4 مرات كما في الخوارزمية التالية:

```
For  4 times  Do
    MOV  BH to  DL
    Shift DL 4 times to Right
    If DL < 10   then
        Convert to character in  0 ......9
    else
        Convert to character in A.........F
    end_if
    Output character
    Rotate  BX  left 4 times
END_For
```

تمارين

1 – قم بإجراء العمليات المنطقية التالية:

a. 10101111 AND   10001011   b. 10110001 OR   01001001

c. 01111100 XOR   11011010   d. Not 01011110

2– ما هي الأوامر المنطقية التي تقوم بالآتي:

أ- وضع الرقم '1' في الخانة ذات الوزن الأكبر والخانة ذات الوزن الأصغر في المسجل BL مع ترك باقي الخانات بدون تغيير.

II- عكس قيمة الخانة ذات الوزن الأكبر في المسجل BX مع ترك باقي الخانات دون تصغير.

III- عكس قيمة كل الخانات الموجودة في المتغير Word1.

3- استخدم الأمر Test في الآتي:

1. وضع الرقم '1' في بيرق الصفر إذا كان المسجل AX يحتوى على الرقم صفر.
2. وضع الرقم '0' في بيرق الصفر إذا كان المسجل DX يحتوى على عدد فردى.
3. وضع الرقم '1' في بيرق الإشارة إذا كان المسجل DX يحتوى على عدد سالب.
4. وضع الرقم '1' في بيرق الصفر إذا كان المسجل DX يحتوى على صفر.
5. وضع الرقم '1' في بيرق خانة التطابق إذا كان المسجل BL يحتوى على عدد زوجي من الخانات التي تحتوى على الرقم '1'

4- إذا كان المسجل AL يحتوى على الرقم 11001011b وكانت قيمة بيرق المحمول تساوى واحد CF=1 ما هي محتويات المسجل AL بعد تنفيذ كل من العمليات التالية

(افترض القيمة الابتدائية مع كل عملية).

a. SHL AL,1 b. SHR AL , 1
c. ROL AL , CL ; if CL contains 2 d. ROR AL, CL ; if CL contains 3
e. SAR AL,CL ; if CL contains 2 f. RCL AL, CL if CL contains 3
g. RCR AL ,CL; if CL contains 3

5- أكتب الأمر أو الأوامر التي تقوم بعمل التالي مفترضاً عدم حدوث فيضان.

أ- مضاعفة الرقم B5h

ب- ضرب محتويات المسجل AL في الرقم 8

جـ- قسمة الرقم 32142 على الرقم 4 ووضع النتيجة في المسجل AX

د- قسمة الرقم 2145- على الرقم 16 ووضع النتيجة في المسجل BX

6- أكتب الأمر أو الأوامر التي تقوم بالآتي:

1. إذا كان المسجل AL يحتوي على رقم أقل من 10 قم بتحويل الرقم الى الحرف المناظر.
2. إذا كان المسجل DL يحتـوي علـى الكـود ASCII لحرف كبير. قم بتحويله لحرف صغير.

7 - أكتب الأمر أو الأوامر التي تقوم بالآتي:

1. ضـرب محتويـات المسـجل BL فـي الـرقم 10D مفترضاً عدم حدوث فيضان.
2. إذا كان المسجل AL يحتوي على عدد موجب. قم بقسمة هذا الرقم على (8) وطرح الباقي في المسجل AH (مساعدة : استخدم الأمر ROR).

تمارين البرامج :

8 - أكتب برنامج يقوم بسؤال المستخدم لإدخال حرف. يقوم البرنامج في السطر الثـاني بطباعـة الكـود الـــ ASCII في الصورة الثنائية للحرف المدخل وكذلك عدد الخانات التي تحتوي على العدد '1' في الكود . مثال

TYPE A CHARACTER : A
THE ASCII CODE OF A IN BINARY IS 01000001
THE NUMBER OF 1 BITS IS 2

9 - أكتب برنامج يقوم بسؤال المستخدم لإدخال حرف. يقوم البرنامج في السطر الثاني بطباعة الكود الـ ASCII في الصورة السداسية عشر للحرف المدخل. يقوم البرنامج بالتكرار حتى يقوم المستخدم بعدم إدخال حرف والضغط على مفتاح الإدخال.

TYPE A CHARACTER : 7
THE ASCII CODE OF 7 IN HEX IS :   37
TYPE A CHARCTER :

10 - أكتب برنامج يقوم بسؤال المستخدم لإدخال عدد سداسي عشر مكون من 4 خانات كحد أقصى. يقوم البرنامج في السطر الثاني بطباعة الرقم المدخل في الصورة الثنائية. إذا قام المستخدم بإدخال قيمة غير مسموح بها (رقم غير سداسي عشري) يقوم البرنامج بسؤاله بالمحاولة مره أخرى.

TYPE A HEX NUMBER   (0000 - FFFF) :
xa
ILLEGAL HEX DIGIT, TRY AGAIN ; 1ABC
IN BIRARY IT IS 0001101010111100

11- اكتب برنامج يقوم بسؤال المستخدم لإدخال رقم ثنائي يكون من 16 خانة لعدد

أقصي.يقوم البرنامج في السطر التالي بطباعة الرقم في الصورة السداسية عشر . إذا

قام المستخدم بإدخال رقم غير ثنائي (يحتوي علــي خانة لا تساوي "0" أو لا تساوي

"1") يقوم البرنامج بسؤال المستخدم ليحــاول مــره أخري.

TYPE A BINARY NUMBER UB TO 16 DIGITS : 112

ILLEGAL BINARY DIGIT , TRY AGAIN : 11100001

IN HEX IT IS EI

12- أكتب برنامج يقوم بسؤال المستخدم لإدخال عددين ثنائيين بطول أقصى 8 خانات

يقوم البرنامج بطباعة مجموع العددين فــي الســطر التالي في الصورة الثنائية أيضاً .

إذا قام المستخدم بإدخال رقم خطأ يتم طلب إدخال الرقم مره أخري.

TYPE A BINARY NUMBER , UP TO 8 DIGITS : 11001010

TYPE A BINARY NUMBER , UP TO 8 DIGITS : 10011100

THE BINARY SUM IS 101100110

13 - أكتب برنامج يقوم بسؤال المســتخدم لإدخــال عدد سداسي عشر بدون إشارة يقوم

البرنامج بطباعة مجموع العددين في السطر التالي .
إذا ادخل المستخدم رقم خطأً

يتم سؤاله للمحاولة مره أخــري . يقــوم البرنــامج
باختبار حدوث عملية الفيضان

بدون إشارة ويطبع النتيجة الصحيحة

TYPE A HEX NUMBER (0 – FFFF) : 21AB

TYPE A HEX NUMBER (0 – FFFF) : FE03

THE SUM IS 11FAE

14- اكتب برنامج يقوم بسؤال المســتخدم بإدخــال
أرقام عشرية تنتهي بالضغط على

مفتاح الإدخال . يقــوم البرنــامج بحســاب وطباعــة
مجموع الخانات العشرية التي تم

إدخالها في السطر التالي في الصــورة السداســية
عشر . إذا قام المستخدم بإدخال

رقم خطأً (لا يقع بين 0 , 9) يقوم البرنامج بسؤاله
للمحاولة مرة أخرى

ENTER A DECIMAL DIGIT STRING : 1299843

THE SUM OF THE DIGITS IN HEX IS : 0024

الفصل السابع
المكدس ومقدمة عن الإجراءات
The Stack and Introduction to Procedures

يتم استخدام مقطع المكدس للتخزين المؤقت للعناوين والبيانات أثناء عمل البرنامج وفى هذا الفصل سنتناول طريقة عمل المكدس واستخدامه في عملية النداء للبرنامج الفرعية Procedures وذلك لتوضيح كيفية وضع قيم في المكدس وأخذ قيم منه باستخدام الأوامر pop, push ثم نتطرق لميكانيكية نداء البرامج الفرعية مع توضيح مثال لذلك.

يعتبر المكدس كمصفوف أحادي في الذاكرة ويتم التعامل مع طرف واحد فقط منه حيث يتم إضافة العنصر في قمة المكدس ويتم أخذ آخر عنصر في عملية السحب التالية بمعني انه يعمل بطريقة آخر مدخل هو أول مخرج LIFO (Last In first out) يجب على كل برنامج أن يقوم بتحديد منطقة في الذاكرة وتعمل كمكدس كما ذكرنا في الفصول السابقة وذلك باستخدام الأمر.

STACK 100h

حيث يشير مسجل مقطع المكدس SS إلى مقطع المكدس في المثال السابق ويحتوى مؤشر المكدس SP على القيمة 100h وهى تشير إلى مكدس خالي وعند وضع قيم فيه يتم إنقاص هذه القيمة.

وضع قيم في المكدس والأوامر PUSH , PUSHF:

يتم استخدام الأمر PUSH لإدخال قيمة في المكدس وصيغته

PUSH SOURCE

حيث المصدر هو مسجل أو موقع في الذاكرة بطول 16 خانة . مثلاً

PUSH AX

ويتم في هذه العملية الآتي:

1- إنقاص قيمة مؤشر المكدس SP بقيمة 2

2- يتم وضع نسخة من المصدر في الذاكرة في العنوان SS:SP

لاحظ أن محتويات المصدر لا يتم تغييرها.

الأمر PUSHF يقوم بدفع محتويات مسجل البيارق في المكدس. فمثلاً لو كانت محتويات مؤشر المسجل SP هى الرقم 100h قبل تنفيذ العملية فبعد تنفيذ الأمر PUSHF يتم إنقاص 2 من محتويات المسجل SP لتصبح قيمته 00FEh بعد

ذلك يتم عمل نسخة من محتويات مسجل البيارق في مقطع المكدس عند الإزاحة 00FE.

سحب قيمة من المكدس والأوامر POP , POPF:

لسحب قيمة من المكدس يتم استخدام الأمر POP وصيغته

POP Destination

حيث المستودع عبارة عن مسجل 16 خانة (ماعدا المسجل IP) أو خانة في الذاكرة مثلاً POP BX

وتنفيذ الأمر POP يتضمن التالي:

1- نسخ محتويات الذاكرة من العنوان SS:SP الى المستودع

2- زيادة قيمة مؤشر المكدس SP بالقيمة 2

الأمر POPF يقوم بسحب أول قيمة من المكدس إلى مسجل البيارق.

لاحظ أن أوامر التعامل مع المكدس لا تؤثر في البيارق كما أنها تتعامل مع متغيرات بطول 16 خانة ولا تتعامل مع 8 خانات. فمثلاً الأمر التالي غير صحيح

Push AL ; ILLEGAL

بالإضافة إلى برنامج المستخدم User Program يقوم نظام التشغيل باستخدام المكدس لأداء عمله فمثلاً عند استخدام نداء المقاطعة INT 21h يقوم نظام التشغيل بتخزين القيم المختلفة للمسجلات في المكدس ثم استرجاعها مره أخرى عند الانتهاء من عمل نداء المقاطعة والعودة للبرنامج وبالتالي لا يتأثر برنامج المستخدم بالتغييرات التي تمت في المسجلات.

مثال لتطبيقات استخدام المكدس:

لان نظرية عمل المكدس تعتمد على أن آخر قيمة تم تخزينها هي أول قيمة سيتم سحبها LIFO ستقوم في هذا الجزء بتوضيح مثال يقوم بقراءة جملة من لوحة المفاتيح. يقوم البرنامج في السطر التالي بطباعة الجملة بصورة عكسية مثال للتنفيذ:

? this is a test
tset a si siht

والخوارزمية هي:

Display '?'
Initialize count to 0
Read a character

```
        While    Character is not a Carriage return
    Do
            push character onto the stack
            increment  counter
            Read a character
        End_While
        Go to New line
        For count times Do
            Pop a character from the stack
            Display  it
        End_For
```

يستخدم البرنامج المسجل CX للاحتفاظ بعدد حروف الجملة التي تم إدخالها بعد الخروج من حلقة while يكون عدد الحروف الموجودة في المسجل CX وتكون كل الحروف التي تم إدخالها موجودة في المكدس بعد ذلك يتأكد البرنامج من انه قد تم إدخال حروف وذلك بالتأكد من أن المسجل CX لا يساوى صفر.

```
.MODEL      SMALL
.STACK      100H
.CODE
MAIN     PROC
            ; display user prompt
    MOV      AH,2
        MOV      DL,'?'
```

```
        INT 21H
        ;initialize character count
        XOR     CX , CX
        ;read character
        MOV     AH , 1
    INT 21H
        ;while character is not a carriage return do
WHILE_:
    CMP     AL , 0DH
        JE   END_WHILE
        PUSH    AX
        INC  CX
        INT 21H
        JMP     WHILE_
END_WHILE:
        MOV     AH , 2
        MOV     DL , 0DH
        INT 21H
    MOV     DL , 0AH
        INT 21H
        JCXZ    EXIT
TOP:
        POP     DX
        INT 21H
        LOOP    TOP
EXIT:   MOV     AH , 4CH
        INT 21H
```

```
MAIN    ENDP
END     MAIN
```

البرامج الفرعية PROCEDURES:

عند كتابة البرنامج وبالذات الكبيرة منها يتم تقسيم البرنامج إلى مجموعة البرامج الفرعية الصغيرة والتي تسهل كتابتها ويكون عمل هذه البرامج الفرعية كوحدة مستقلة لها مدخلات وتؤدى وظيفة محدودة ولها مخرجات محدده وواضحة وبالتالي يسهل استعمالها وكذلك إعادة استخدامها في برامج أخرى كما سنرى فيما بعد.

وبالتالي فان طريقة كتابة البرامج تبدأ بتقسيم المشكلة إلى مجموعة من البرامج الصغيرة ثم توزيع هذه البرامج الصغيرة وكتابة كل منها على حده واختباره وبعد ذلك يتم تجميع هذه البرامج الصغيرة لتعطى برنامج كبير.

أحد هذه البرامج الصغيرة هو البرنامج الرئيسي وهو يعتبر نقطة الدخول للبرنامج ويقوم بدوره بنداء البرامج الفرعية الأخرى والتي يقوم كل منها بدوره بعد الانتهاء بالعودة إلى البرنامج الذي قام باستدعائه.

وفى حالة البرامج ذات المستوى العالي High Level

Programming Languages تكـــون عمليـــة النـــداء والعودة مخفية عن المبرمج ولكن فـــي لغـــة التجميـــع يجب كتابة أمر الاستدعاء CALL أمـــر العـــودة RET كما سنرى عند التعامل مع البرامج الفرعية.

التصريح عن البرامج الفرعية Procedure Declaration:

يتم التصريح عن البرنامج الفرعي على النحو التالي:

```
Name    PROC    type
        ; Body of the procedure
        RET
Name    ENDP
```

حيث Name هـــو اســـم الإجـــراء و type هـــو معامـــل Operand اختياري ويأخذ الصـــيغتين NEAR أو FAR حيث NEAR تعنى أن نداء البرنامج الفرعي يتم مـــن داخل نفس المقطع أما FAR فتعنى إن نداء البرنامج الفرعي يتم من مقطع مختلف. وإذا لم يتم كتابة شئ يتم افتراض أن البرنامج الفرعي من النوع NEAR.

الأمـــر RET (Return) يـــؤدى إلـــى إنهـــاء البرنـــامج الفرعي والعودة إلى البرنامج الذي قام باســـتدعائه . وأي برنامج فرعى يجب أن يقـــوم باســـتخدام الأمـــر RET للعودة إلى البرنامج الذي قام استدعاؤه (فيمـــا عـــدا

البرنامج الرئيسي) ويتم هذا عادة في آخر جملة في البرنامج الفرعي.

الاتصال بين البرامج الفرعية

يجب على أي برنامج فرعى أن تكون له إمكانية استقبال المدخلات إليه وان يقوم بإعادة النتيجة إلى البرنامج الذي قام بندائه إذا كان عدد المدخلات والمخرجات صغير يمكن استخدام المسجلات كأماكن يتم عن طريقها الاتصال بين البرامج الفرعية المختلفة أما إذا كان عدد المدخلات أو المخرجات كبير نضطر إلى استخدام طرق أخري سيتم مناقشتها في الفصول التالية.

توثيق البرامج الفرعية

يجب بعد الانتهاء من كتابة البرنامج الفرعي القيام بعملية التوثيق الكامل له حتى يسهل في أي وقت

وبواسطة أي شخص استخدام هذا البرنامج الفرعي إذا أراد ذلك ويشمل التوثيق على:

1- الشرح العام للوظيفة التي يقوم بها البرنامج الفرعي

2- المدخلات: يتم فيها تعريف المدخلات المختلفة للبرنامج الفرعي

3- المخرجات: يتم فيها تعريف المخرجات المختلفة للبرنامج الفرعي

4- الاستخدامات يتم توضيح البرامج الفرعية (إن وجدت) والتي يقوم هذا البرنامج الفرعي باستخدامها.

__الامر RET , CALL:__

لنداء برنامج يتم استخدام الأمر CALL وله صيغتين الأولي مباشر DIRECT وهى على النحو التالي

CALL name

حيث name هو اسم البرنامج الفرعي المطلوب نداؤه.

والصيغة الثانية للنداء الغير مباشر Indirect وهى على الصورة

CALL address_expression

حيث CALL address - expression تحدد المسجل أو المتغير الذي يحوى عنوان البرنامج الفرعي المطلوب تنفيذه.

عند نداء برنامج فرعى يتم الآتي

1- يتم تخزين عنوان الرجوع Return address في المكدس وهو الأمر التالي

للأمر CALL في البرنامج الذي قام بالنداء

2- يتم وضع عنوان إزاحة أول أمر في البرنامج الفرعي في المسجل

التعليمات IP وبالتالي يتم التفرع إلى ذلك البرنامج الفرعي

وللعودة من أي برنامج فرعي نستخدم الأمر RET حيث تؤدى إلى اخذ عنوان الرجوع من المكدس ووضعه فى مسجل التعليمات مما يؤدى إلى العودة للبرنامج الذي قام بالنداء

ويمكن ان يأخذ الصورة RET Pop_value

حيث Pop_value معامل اختياري. إذا كانت Pop_value = N فان معنى ذلك أن يتم سحب عدد N -Bytes إضافية من المكدس.

<u>مثال لبرنامج فرعى:-</u>

سنوضح هنا مثال لبرنامج فرعي يتم فيه حساب حاصل ضرب رقمين موجبين a,b وذلك باستخدام عملية الجمع والإزاحة وتكون خوارزمية الضرب على النحو التالي :-

```
Product = 0
Repeat
      If LSB of B is 1   then
            Product = Product + A
      End_if
      Shift left  A
      Shift right  B
until B = 0
```

ولمتابعة الخوارزمية اعتبر ان A= 111b و1101b = B وبتطبيق الخوارزمية نجد ان

```
product = 0
since LSB of B is 1   ,  product = 0 + 111b = 111b
shift left A:                A = 1110b
shift right B :              B = 110b
since LSB of B is 0 ;
shift left A :               A=11100b
shift right B :              B = 11b
since LSB of B is 1 ;        product = 111b + 11100b = 100011b
shift left A :               A = 111000b
shift right B :              B  =1b
```

since LSB of B is 1 , product = 100011b + 111000b = 1011011b
shift left A : A = 1110000
shift right B : B = 0
since LSB of B is 0 ,
return Product = 1011011b = 91d

وفيما يلي البرنامج:

```
.MODEL      SMALL
.STACK      100H
.CODE
MAIN        PROC
        CALL        MULTIPLY
        MOV     AH,4CH
        INT     21H
MAIN    ENDP
MULTIPLY  PROC
        PUSH    AX
        PUSH    BX
        XOR     DX , DX
REPEAT:
        TEST    BX , 1
        JZ          END_IF
        ADD     DX , AX
END_IF:
        SHL     AX , 1
        SHR     BX , 1
        JNZ     REPEAT
```

```
        POP    BX
        POP    AX
        RET
MULTIPLY  ENDP
END     MAIN
```

هنا يقوم الإجراء باستقبال المدخلات في المسجلين AX و BX ويتم حساب حاصل الضرب في المسجل DX. وتجنبنا لحدوث الفيضان يحتوى المسجلان AX و BX على رقمين أقل من FFh.

يبدأ دائماً أي برنامج فرعى بتخـزين قـيم المسـجلات التي سيقوم باستخدامها في المكدس باستخدام مجموعـه من أوامر PUSH ثم بعد انتهاء عمـل الإجـراء يـتم استرجاع القيم القديمة من المكدس باستخدام مجموعـة من أوامر pop وذلك فيما عدا المسجلات التي يقـوم بإرجاع النتيجة فيها وذلك حتى لا يتم تغيير المسـجلات للبرنامج الأصلي وبالتالي فـان الشـكل العـام للبـرامج الفرعية هو:

```
    NAME   PROC
        Push    AX
        Push    BX
        الأوامر داخل الإجراء :
        Pop     BX
        Pop     AX
```

```
RET
NAME    ENDP
```

تمارين:

1- إذا كان تعريف المكدس في البرنامج هو
.STACK 100H

أ- ما هي محتويات مؤشر المكدس SP بعد بداية تنفيذ البرنامج مباشرة؟

ب- افترض أن المسجلات التالية تحتوى على القيم الموضحة

AX = 1234h , BX = 5678h , CX = 9ABCh , and SP=100h

وضع محتويات المسجلات AX , BX , CX , SP بعد تنفيذ الجزء التالي البرنامج

```
PUSH    AX
PUSH    BX
XCHG    AX , CX
POP     CX
PUSH    AX
POP     BX
```

3- عندما يمتلئ المكدس تكون محتويات مؤشر المكدس هل الرقم صفر (SP=0). اذا تم وضع كلمة جديدة في المكدس. ماذا سيحدث للمسجل SP ؟ وماذا يمكن

أن يحدث للبرنامج.

4- افترض أن برنامج به الجزء التالي:

```
CALL    PROC1
MOV     AX , BX
```

افترض أن:

أ- الأمر MOV AX,BX يقع في الذاكرة في العنوان 08FD:0203

ب- البرنامج PROC1 من النوع Near ويقع في العنوان 08FD:300h

ج- يحتوي مؤشر المكدس على القيمة SP = 010Ah

ما هي محتويات المسجلين SP , IP بعد تنفيذ الأمر CALL PROC1 مباشر وما هي الكلمة الموجودة في قمة المكدس.

5- اكتب برنامج يقوم بكل الآتي:

أ- وضع الكلمة الموجودة في قمة المكدس في المسجل AX دون تغيير محتويات المكدس.

ب- وضع الكلمة الثانية في المكدس في المسجل CX بدون تغيير محتويات المكدس.

جـ - استبدال محتويات الكلمة الأولى في المكدس مع الكلمة الثانية

6 - في المعادلات الجبرية يمكن استخدام الأقواس لتوضيح عملية محددة وتحديد أولويات الحساب

حيث نستخدم الأقواس ' [ ] { } ( ) ' وتنتهي المعادلة بالضغط علي مفتاح الإدخال. للتأكد

من صحة وجود الأقواس يجب أن يكون نوع كل قوس من نفس نوع آخر قوس تم فتحه.

فمثلاً المعادلة التالية صحيحة

(A + { B - (D - E) + [A + B] })

بينما المعادلة التالية غير صحيحة

(A + { B - C])

يمكن التأكد من المعادلة باستخدام المكدس حيث نقوم بقراءة المعادلة من اليسار وكلما

وجدنا قوس جديد يتم إدخاله في المكدس. إذا كان القوس هو قوس إغلاق يتم مقارنته مع

آخر قوس في المكدس بعد إخراجه منه فإذا كانا من نفس النوع نواصل القراءة وإذا لم يكن

من نفس النوع يعني ذلك أن المعادلة خطأ. في النهاية إذا تم تفريغ كل الأقواس من المكدس

تكون المعادلة صحيحة وإذا ظلت هناك أقواس في المكدس تكون المعادلة غير صحيحة.

أكتب برنامج يقوم بقراءة معادلة تحتوي على الأنواع الثلاثة من الأقواس المذكورة. يستمر البرنامج الإدخال حتى تنتهي المعادلة أو يقوم المستخدم بإدخال معادلة خطأ حيث يقوم البرنامج في هذه الحالة بإخطار المستخدم بأن المعادلة خطأ.

7 – نستخدم الطريقة التالية لتوليد أرقام عشوائية في المدى من 1 إلى 32767

- ابدأ بأي رقم.
- قم بإزاحة الرقم لليسار خانة واحدة.
- استبدل الخانة رقم صفر بالخانتين 14 و 15 بعد عمل XOR لهما.
- قم بوضع الرقم صفر في الخانة 15.

المطلوب كتابة الإجراءات التالية:

أ – إجراء يسمي READ وهو يقرأ رقم ثنائي من المستخدم ويقوم بتخزينه في المسجل BX

ب – إجراء يسمي RANDOM وهو يستقبل عدد في المسجل BX ويقوم بإعادة رقم عشوائي حسب الخوارزمية المذكورة

جـ – إجراء يسمي WRITE وهو يقوم بطباعة محتويات المسجل BX في الصورة الثنائية.

أكتب برنامج يقوم بطباعة علامة الاستفهام '?' ثم يقوم بنداء الإجراء READ لقراءة رقم ثنائي ثم نداء الإجراء RANDOM لحساب الرقم العشوائي ثم نداء الإجراء WRITE لحساب

وطباعة 100 رقم عشوائي بحيث يتم طباعة 4 أرقام فقط في السطر الواحد مع 4 فراغات تفصل بين الأعداد.

الفصل الثامن

أوامر الضرب والقسمة

Multiplication and Division Instructions

رأينا في الأجزاء السابقة عملية الضرب والقسمة على الرقم اثنين ومضاعفاته باستخدام أوامر الإزاحة لليسار ولليمين. في هذا الفصل سنقوم بتوضيح

العمليات التي تقوم بعمليات الضرب والقسمة على أعداد غير العدد اثنين ومضاعفاته.

تختلف عمليات الضرب للأرقام بإشارة منها في حالة الأرقام بـدون إشـارة وكـذلك عمليـات القسـمة وبالتالي لدينا نوعين من أوامر الضرب والقسـمة أحدهما للأرقام بإشـارة والأخـرى للأرقـام بـدون إشارة وكذلك هناك صور للتعامل مع أرقام بطول 8 خانات فقط وأخرى للتعامل مع أرقام بطول 16 خانه.

أحد اسـتخدامات أوامـر الضـرب والقسـمة هـو استخدامها لإدخال وإخراج الأرقـام فـي الصـورة العشرية مما يزيد من كفاءة برامجنا.

عمليات الضرب MUL & IMUL

نبدأ مناقشة عمليات الضرب بالتفرقة بين الضـرب بإشارة والضرب بدون إشارة فعلى سـبيل المثـال إذا تم ضـرب الـرقمين الثنـائيين 10000000 و 11111111 فلدينا هنا تفسيرين للرقمين. التفسـير الأول هو أن الأرقام ممثله بدون إشارة وبالتـالي فإن المطلوب هو ضرب الـرقم 128 فـي الـرقم 255 ليصبح الناتج 32644. أما التفسير الثاني هـو

أن الأرقام عبارة عن أرقام بإشارة فإن المطلوب هو ضرب الرقم 128- في الرقم 1- لتصبح النتيجة 128 وهى نتيجة مختلفة تماماً عن النتيجة التي تم الحصول عليها في التفسير الأول (32640).

لأن عمليات الضرب للأرقام بإشارة تختلف عن عمليات الضرب للأرقام بدون إشارة يتم استخدام أمرين: الأول يستخدم في عمليات الضرب للأرقام بدون إشارة وهو الأمر MUL (Multiply). والثاني يستخدم في عمليات الضرب للأرقام بإشارة وهو IMUL (Integer Multiply). تقوم هذه الأوامر بعملية الضرب لرقمين بطول 8 خانات ثنائية ليكون حاصل الضرب بطول 16 خانه ثنائية أو لضرب رقمين بطول 16 خانه ثنائية ليكون حاصل الضرب بطول 32 خانه ثنائية. والصيغة العامة للأمرين هي:

```
    MUL     Source
&   IMUL    Source
```

هنالك صورتان للتعامل مع هذه الأوامر الأولى عند ضرب أرقام بطول 8 خانات والثانية عند ضرب أرقام بطول 16 خانه

استخدام أرقام بطول 8 خانات Byte Form

حيث يتم ضرب الرقم الموجود فـي المسـجل AL في الرقم الموجود في المصدر Source وهو إما محتويات مسجل أو موقع في الذاكرة (غير مسموح باستخدام ثوابت). يتم تخزين النتيجـة (بطـول 16 خانه) في المسجل AX.

استخدام أرقام بطول 16 خانات Word form

في هذه الصورة يتم ضرب الرقم الموجـود فـي المسجل AX في الرقم الموجود في المصدر وهـو إما مسجل أو موقع في الـذاكرة (غيـر مسـموح باستخدام ثوابت). يتم تخزين النتيجة (32 خانه) في المسجلين DX , AX بحيث يحوى AX على النصـف السفلي و DX على النصف العلوي وتكتب النتيجة عاده علـى الصـورة DX:AX. { النصـف السـفلي : النصف العلوي }

في حالة ضرب الأرقام الموجبة نحصل على نفس النتيجة عند استخدام الأمرين IMUL, MUL.

<u>تأثر البيارق بأوامر الضرب</u>

لا تتأثر بأوامر الضرب كل من البيارق SF, ZF, AF, PF

<u>أما بالنسبة للبيرقين Cf/Of :</u>

أ/ في حالة استخدام الأمر MUL

تأخذ البيارق القيمـة (0) (CF/OF = 0) إذا كان النصف العلوي من النتيجة يساوي صفر وتأخذ البيارق القيمـة (1) إذا لم يحدث ذلك.

ب/ في حالة استخدام الأمر IMUL

يأخذ البيرق القيمـة 0 (CF/OF = 0) إذا النصف العلوي هو عبارة عن امتداد لإشارة النصف السفلي Sign Extension (أي أن كل خانات النصف العلوي تساوي خانه الإشارة MSB من النصف السفلي) وتأخذ البيارق القيمة (1) (CF/OF = 1) إذا لم يحدث ذلك.

بالنسبة للأمرين نلاحظ أن البيارق CF/OF تأخذ القيمة (1) اذا كانت النتيجة كبيره ولا يمكن تخزينها في النصف السفلي فقط (AL في حالة ضرب رقمين بطول 8 خانات و AX في حالة ضرب رقمين بطول 16 خانه). وبالتالي يجب التعامل مع باقي النتيجة والموجود في النصف العلوي.

أمثلة:

في هذا الجزء سنقوم باستعراض بعض الأمثلة لتوضيح عمليات الضرب المختلفة.

1/ اذا كان BX = ffffh , AX = 1

الأمر	النتيجة بالعشري	النتيجة (سداسي عشري)	AX	DX	CF/OF
MUL BX	65535	0000ffff	ffff	0000	0
IMUL BX	-1	Ffffffff	Ffff	ffff	0

2/ اذا كان BX = ffffh, AX=ffffh

الأمر	النتيجة (عشري)	النتيجة (سداسي عشر)	AX	DX	CF/OF
MUL BX	4294836225	FFFE0001	0001	FFFE	1
IMUL BX	1	00000001	0001	0000	0

3/ اذا كان AX = 0fffh

الأمر	النتيجة (عشري)	النتيجة (سداسي عشر)	AX	DX	CF/OF
MUL AX	16769025	00ff E00l	Eoo1	00ff	1
IMUL AX	16769025	00ff E00l	E001	00ff	1

4/ اذا كان CX = ffffh , AX = 0100h

الأمر	النتيجة (عشري)	النتيجة (سداسي عشر)	AX	DX	CF/OF
MUL CX	16776960	00FFFF00	FF00	00FF	1
IMUL CX	-256	FFFFFF00	FF00	FFFF	0

تطبيقات بسيطة على أوامر الضرب:

1/ حساب معادلات مختلفة فمثلًا إذا أردنا حساب المعادلة التالية

A = 5 x A - 12 x B

نقوم بالآتي بافتراض عدم حدوث فيضان

```
MOV     AX,5        ;  AX =  5
IMUL    A           ;  AX = 5 *A
MOV     A , AX      ;  A = 5 *A
MOV     AX,12       ;  AX = 12
```

```
IMUL     B              ;  AX = 12 x B

SUB          A ,AX        ;  A = 5 x A - 12 x B
```

2/ حساب مضروب عدد

المطلوب هنا كتابة إجراء PROCEDURE يسمى FACTORIAL يقوم هذا الإجراء بحساب N! لأي عدد صحيح موجب (N) يستلم الإجراء العدد الصحيح N في المسجل CX ويقوم الإجراء بإعادة مضروب N في المسجل AX. (نفترض عدم حدوث فيضان)

<u>تعريف مضروب العدد هو:</u>

$$N\,! = 1 \quad \text{if} \quad N = 1 \quad \text{Then}$$

$$N\,! = N \times (N-1) \times (N-2) \times ..... \times 2 \times 1 \quad \text{if} \quad N > 1 \quad \text{Then}$$

ويتم ذلك حسب الخوارزمية التالية

```
PRODUCT = 1
 Term = N
 For  N Times Do
      product = product * term
      Term = Term -1
 END_For
```

ويصبح الإجراء على الصورة التالية:

```
FACTORIAL PROC
          ; Computes N1
          MOV     AX, 1
     Top:      Mul CX
          Loop Top
          RET
FACTORIAL ENDP
```

لاحظ هنا أن هذا الإجراء يقوم بحساب مضروب الأعداد التي لا يتعدى مضروبها 65535 حيث لا يتم التعامل مع حالات الفيضان.

أوامر القسمة DIV , IDIV

كما في حالة عمليات الضرب فان عمليات القسمة تختلف عند التعامل مع الأرقام بإشارة عنها في حالة الأرقام بدون إشارة وعلى ذلك نستخدم

في حالة الأرقام بدون إشارة الأمر DIV (Divide)

في حالة الأرقام بإشارة الأمر IDIV (Integer Divide)

والصيغة اللغوية للأمرين كالآتي :

```
DIV Source
IDIVSource
```

عند إجراء عملية القسمة نحصل على خارج القسمة في مسجل وباقي عملية القسمة في مسجل آخر.

لدينا صورتين عند استخدام عملية القسمة إما تستخدم أرقام بطول 8 خانات أو أرقام بطول 16 خانة كما يلي:

استخدام أرقام بطول 8 خانات Byte form

في هذه الصورة تتم قسمة الرقم الموجود في المسجل AX على المصدر ويتم تخزين خارج القسمة (8 بت) في المسجل AL وباقي القسمة (8 بت) في المسجل AH.

استخدام أرقام بطول 16 خانة Word form

في هذه الصورة يتم قسمة الرقم الموجود في المسجلين DX , AX (على الصورة DX :AX حيث DX به النصف العلوي و AX جهة النصف السفلي) على المصدر ويتم تخزين خارج القسمة في المسجل AX وباقي القسمة في المسجل DX.

في حالة الأرقام بإشارة تكون إشارة الباقي هي نفس إشارة الرقم المقسوم. وإذا كان الرقم المقسوم والمقسوم عليه موجبين تكون النتيجة واحدة عند استخدام IDiv , Div.

بعد تنفيذ أوامر القسمة تكون البيارق كلها غير معرفه.

فيضان القسمة Divide Overflow

يتم الفيضان في عملية القسمة إذا كان خارج القسمة رقم كبير لا يمكن تخزينه في المسجل المخصص لذلك. ويتم ذلك عند قسمة رقم كبير جداً على رقم صغير جداً. في هذه الحالة يقوم البرنامج بالانتهاء ويقوم النظام بطباعة رسالة تفيد بحدوث فيضان قسمة " Divide Overflow" .

مثال: إذا كان BX = 0002 , A x =0005 , DX = 0000

الأمر	خارج القسمة (عشري)	باقي القسمة (عشري)	AX	DX
Div BX	2	1	0002	0001
IDIV	2	1	0002	000

1				BX

مثال: إذا كان BX = FFFEh , AX = 0005 , DX = 0000

DX	AX	باقي القسمة (عشري)	خارج القسمة (عشري)	الأمر
0005	0000	5	0	Div B x
0001	FffE	1	-2	Idiv B x

مثال: إذا كان BX = 0002h , AX = fffbh , DX = ffffh

DX	AX	باقي القسمة (عشري)	خارج القسمة (عشري)	الأمر
فيضان عند قسمة الرقم fffffffbh علي2 الناتج (7ffffffeh) لايمكن تخزينه في AX				Div B x
Ffff	FffE	1-	-2	Idiv B x

مثال: BL = Ffh , AX = 00fBh

AH	AL	باقي القسمة (عشري)	خارج القسمة (عشري)	الأمر

FB	0	251	0	Div B L
لان خارج القسمة (يساوى - Divide overflow 25) لا يمكن تخزينه في AL				Idiv B L

تمديد إشارة المقسوم Sign Extension of Dividend

1/ في حالة استخدام أرقام بطول 16 خانة

يكون المقسوم موجـود فـي المسـجلين DX , AX حتى ولو كان الرقم يمكـن تخزينـه فقـط فـي المسجل AX وعلى هذا فـان المسـجل DX يجـب تجهيزه على النحو التالي:

1. عند استخدام الأمر Div يتم وضع الـرقم 0 في المسجل DX
2. عند استخدام الأمر IDIV يجب أن تكون كل الخانات في المسجل DX بنفس قيمـة خانة الإشارة في المسجل AX (أي لو كان الرقم في AX موجب يتم وضع الـرقم 0 في المسجل DX ولو كان الرقم في AX سالب يتم وضع الرقم ffffh فـي المسـجل DX) ولعمل ذلك نسـتعمل الأمـر CWD (convert word to Double word). وبالمثـل لتمديد إشارة AL إلـى AH نسـتعمل الأمـر CBW (Convert Byte to Word)

مثال: اقسم 1250 - على 7

```
MOV     AX , -1250
CWD                      ; prepare DX
MOV     BX , 7
IDIVBX
```

إدخال وإخراج الأرقام العشرية:

رغم أن تمثيل كل الأرقام داخل الكمبيوتر يتم علي صورة أرقام ثنائية إلا أن التعامل مع العالم الخارجي يفضل أن يتم بأرقام في الصورة العشرية وسنتناول في هذا الجزء كيفية قراءة الأرقام بالصورة العشرية وكيفية طباعتها في الشاشة في صورة عشرية.

في الإدخال وعند كتابة رقم في لوحة المفاتيح فان البرنامج يستقبل المدخلات علي أنها سلسلة حروف وبالتالي يجب أولا تحويل الحروف للأرقام الثنائية المناظرة للرقم الذي تم إدخاله. وكذلك في حالة الإخراج حيث يتم تحويل الرقم الثنائي إلى الحروف المناظرة في النظام العشري وطباعتها في الشاشة .

طباعة الأرقام العشرية Decimal Output

ستقوم هنا بكتابة أجراء يسمى outdec وذلك لطباعة محتويات المسجل AX ، إذا احتوي المسجل AX علي رقم

سالب سنقوم بطباعة علامة (-) ثم يتم استبدال المسجل AX بالقيمة
AX- (حيث يحتوى الآن AX علي قيمة موجبة) وبالتالي تحويل العملية لطباعة محتويات المسجل AX والذي يحوى قيمة موجب علي الشاشة في الصورة العشرية وهذه هي الخوارزمية .

1- If AX < 0
2 - print a minus sign
3- Replace AX By its two's complement
4- End-if
5- Get the digits in AX's decimal representation
6- Convert these digits to characters and print them

سنقوم الآن بتوضيح الخطوة 5 في الخوارزمية حيث إذا كان بالمسجل AX رقم ثنائي يناظر الرقم 3567 بالنظام العشري وبطباعة هذا الرقم في الشاشة يقوم بالآتي

اقسم 3567 علي 10 ينتج 356 والباقي 7
اقسم 356 علي 10 ينتج 35 والباقي 6
اقسم 35 علي 10 ينتج 3 والباقي 5

وعلي هذا فان الخانات المطلوبة طباعتهـا هـي باقي القسمة علي الرقم 10 في كـل مـرة ولكـن ترتيبهـا معكوس ولحل هذه المشكلة يتم تخزينها في المكـدس stack ويتم الاحتفاظ بعددها في مسجل محـدد count وهذه هي الخوارزمية .

```
count  =  0
Repeat
     Divide    quotient   by    10
     Push   remainder  on the  stack
     count  =  count + 1
Until quotient  =  0
```

حيث القيمة الابتدائية لخارج القسمة (quotient) هـي الرقم الموجود في المسجل AX وبذلك نوضح الخطوة 6 في الخوارزمية وفيها يتم سحب الأرقام التي تم وضـعها في المكدس (عددها هو موجود في المتغيـر count) وبعد سحب كل رقم تتم طباعتها في الشاشة .

وذلك حسب الخوارزمية التالية

```
For    count    times  do
          Pop a digit  from  the  stack
          Convert    it  to  a character
          Output  the  character
     End_For
```

وعلى هذا يصبح الإجراء كاملا بلغة التجميع علـــي النحــو التالي :

```
OUTDEC  PROC
; Prints AX as a signed decimal integer
; input : AX
; Output : None
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        ;if AX < 0
        OR      AX , AX
        JGE     @END_IF1
        ;Then
        PUSH    AX
        MOV     DL , '-'
        MOV     AH,2
        INT     21H
        POP     AX
        NEG     AX
@END_IF1:
        XOR     CX , CX  ;Get Decimal Digit
        MOV     BX , 10D
@REPEAT1:
        XOR     DX , DX
        DIV     BX
```

```
    PUSH   DX
    INC    CX
    OR     AX , AX
    JNE    @REPEAT1
    ;Convert Digits to characters and print them
    MOV    AH , 2
@PRINT_LOOP:
    POP    DX
    OR     DL , 30H
    INT    21H
    LOOP   @PRINT_LOOP
    POP    DX
    POP    CX
    POP    BX
    POP    AX
    RET
OUTDEC  ENDP
```

يمكننا كتابة الإجراء outdec السابق في ملف مختلف تماما عن الملف الذي يحوى البرنامج الذي سيقوم بهذا الإجراء . وفي ذلك الملف يمكننا استدعاء الإجراء outdec ولكن بعد أن يتم أخطار الـ Assembler بأن هناك إجراءات موجودة في ملف آخر ويتم ذلك باستخدام الإيعاز Include وهو يأخذ الصورة.

Include Filespec

حيث Filespec هو اسم الملف الذي يحوى الإجراء. وعلى ذلك يقوم الـ

Assembler يفتح ذلك الملف والبحث عن الإجراء المطلوب بداخله.

فمثلاً إذا تم حفظ الإجراء OUTDEC السابق في ملف أسميناه PRocfile.ASM يمكن نداء الإجراء من برنامج على النحو التالي:

```
.MODEL       SMALL
.STACK       100h
.CODE
MAIN    PROC
        MOV     AX , 1234
        CALL    OUTDEC
        MOV     AH, 4Ch
        INT 21h
MAIN  ENDP
INCLUDE   PROCFILE.ASM
END   Main
```

قراءة الأرقام العشرية Decimal Input

لقراءة الأرقام العشرية نحتاج لتحويل الحروف ASCII لكل حرف الى القيمة الثنائية المناظر للخانة العشرية وتجميع هذه القيم في سجل. وسنقوم بتوضيح خوارزمية البرنامج.

Total = 0

Read an ASCII Digit

```
Repeat
    Convert character to a Binary value
    Total  = total* l0 + value
    Read a character
Until character  is a carriage return
```

فمثلاً إذا كانت المدخلات هي الرقم 157 سيكون تنفيذ الخوارزمية على النحو التالي:

```
Total  =  0
Read "1"
Convert "1" to 1
Total = 10 x 0 + 1 = 1
Read "5"
Convent "5" to "5"
Total  = 1 x 10 + 5  = 15
Read  "7"
Convent "7" to 7
Total  = 15 x 10 + 7  = 157
```

سنقوم ألان بتطوير الخوارزمية السابقة ووضعها في إجراء يسمى INDEC يقوم الإجـراء بطباعـة علامـة الاستفهام ثم قراءة رقم عشري من لوحة المفـاتيح. قد يبدأ الرقم بإشارة - أو +. إذا احتوى الـرقم على خانة غير عشرية (حرف لا يقع بين 0 و 9) يقوم البرنامج بالقراءة من جديـد. ينتهـي الـرقم بالضغط على مفتاح الإدخال.

```
Print  "?"
Total  =  0
Negative  =  False
Read a character
Case character of
     "- "  :     Negative  =  True
                 Read a character
     "+"   :     Read a character
End_Case
Repeat
     if character is not between "0" and "9"
then
          GO TO Beginning
     Else
          convert character to a Binary
     value
          total  = 10 * total + value
     End if
     Read a character
Until character is a carriage return
IF negative  =  True then
     Total  = -total
End_if
```

ويصبح البرنامج بلغة التجميع كالآتي :

```
INDEC  PROC
; Reads a number in range -32768 to 32767
; input : None
```

```
; Output : AX = Binary equivalent Of Number
        PUSH    BX
        PUSH    CX
        PUSH    DX
@BEGIN: MOV     AH , 2
        MOV     DL , '?'
        INT     21H
        XOR     BX , BX ; total =0
        XOR     CX , CX
        ;Read A Character
        MOV     AH , 1
        INT     21H
        ;Case Char of
        CMP     AL ,'-'
        JE      @MINUS
        CMP     AL , '+'
        JE      @PLUS
        JMP     @REPEAT2
@MINUS: MOV     CX , 1
@PLUS:  INT     21H
@REPEAT2:;If Character Between 0 AND 9
        CMP     AL , '0'
        JNGE    @NOT_DIGIT
        CMP     AL , '9'
        JNLE    @NOT_DIGIT
        ; Convert Character To Digit
        AND     AX ,000FH
```

```
      PUSH    AX
      ;  TOTAL = TOTAL * 10 + DIGIT
      MOV     AX , 10          ;Get 10
      MUL     BX               ;AX = TOTAL * 10
      POP     BX               ;RETRIEVE DIGIT
      ADD     BX , AX          ; TOTAL =
TOTAL*10+DIGIT
      ;Read A Character
      MOV     AH , 1
      INT     21H
      CMP     AL,0DH
      JNE     @REPEAT2
      MOV     AX , BX
      OR      CX , CX
      JE      @EXIT
      NEG     AX
@EXIT:  POP     DX
      POP     CX
      POP     BX
      RET
@NOT_DIGIT:
      MOV     AH , 2
      MOV     DL , 0DH
      INT     21H
      MOV     DL , 0AH
      INT     21H
      JMP     @BEGIN
```

```
INDEC  ENDP
```

الآن ولاختبار الإجــراء يــتم وضــعه فــي الملــف procfile. ASM مــع الاجــراء OutDec ثــم نقــوم بكتابة البرنامج الرئيس بحيث يقوم بنداء الإجراءين على النحو التالي حيث يتم نداء الإجــراء INDEC لقراءة رقم عشري وإعادته في المسجل AX بعدها مباشرة يتم نداء الإجراء OUTdec لطباعــة الــرقم الموجود في المسجل AX في الصورة العشرية على الشاشة.

```
TITLE DECIMAL: READ AND WRITE A
DECIMAL NUMBER
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
     ;INPUT A NUMBER
     CALL          INDEC
     PUSH    AX
     ;MOVE CURSOR TO NEXT LINE
     MOV       AH , 2
     MOV       DL , 0DH
```

```
        INT     21H
        MOV     DL , 0AH
        INT     21H
        ;OUTPUT A NUMBER
        POP     AX
        CALL    OUTDEC
        ;EXIT
        MOV         AH,4CH
        INT     21H
MAIN                ENDP
INCLUDE             PROCFILE.ASM
END     MAIN
```

الفيضان Overflow

يقوم الإجراء Indec بالتعامل مع الأرقام الخطأ (التي تحتوى على خانة غير عشرية) ولكن لا يتعامل مع الأرقام الكبيرة والتي لا يستطيع المسجل AX أن يسعها (الأرقام خارج المدى - 32768 إلى 32767). وإذا كان الرقم خارج هذا المدى يحدث فيضان إدخال Input Overflow.

وقد يحدث هذا الفيضان عند تنفيذ أمرين:

الأول عند ضرب المتغير total في 10

والثاني عند جمع القيمة الجديدة للمتغير total.

ولتوضيح الحالة الأولي قد يقوم المستخدم بإدخال الرقم 99999 حيث يحدث الفيضان عند ضرب الرقم 9999 في 10 أما الحالة الثانية إذا ادخل المستخدم الرقم 32769 يحدث الفيضان عند جمع الرقم 9 إلى الرقم 32760 ويمكن التأكد من ذلك وتعديل الخوارزمية لتصبح على الصورة التالية.

```
Print "?"
Total  = 0
Negative  =  false
Read a character
case character of
      "-"   :     Negative  =  True
                  Read a character
      "+"   :     Read a character
End_Case
Repeat
      If character is not between " 0 " & " 9 "
then
            GO TO Beginning
```

```
        Else
            Convert character to a value
            Total  = 10 x total
            If overflow then
                go to Beginning
            Else
                Total  = total + value
                If overflow then
                    Go To Beginning
                End_If
            End_If
        endif
        Read a character
    Until character is a carriage return
    If Negative  =  True then
        Total  =  -  total
    End_if
```

تمارين:

1/ وضع محتويات المسجلين DX , AX وكذلك البيارق CF/OF بعد تنفيذ كل من الاتى:

أ/ الأمر MUL BX إذا كان BX = 0003h , AX = 0008h

ب/ الأمر MUL BX إذا كان BX = 1000h , AX = 00ffh

ج/ IMUL CX إذا كان CX = FFFFh , AX = 0005h

د/ MOL word إذا كان word = FFFFh , AX = 8000h

هـ/ MUL 10h إذا كان AX = FFE0h

2/ وضع محتويات المسجل AX والبيارق Cf/of بعد تنفيذ كل من الأوامر التالية:

أ/ الأمر MUL BL اذا كان BL = 10h , AL = ABh

ب/ الأمر TMUL BL اذا كان BL = 10h , AL = ABh

ج/ الأمر MUL Ah اذا كان AX = 01ABh

د/ الأمر IMUL Byte1 اذا كان Byte1 = Fbh , AL = 02h

3/ وضع محتويات المسجلين AX , DX عند تنفيذ الاوامر التالية أو وضع حدوث فيضان:

أ/ الأمر Div BX إذا كان B x = 0002h, AX = 0007 , DX = 0000h

ب/ الأمر Div BX إذا كان B x = 0010h , AX = FFFEh , DX = 0000h

ج/ الأمر IDIV BX إذا كان BX = 0003h , AX = fffch , DX = ffffh

د/ الأمر Div BX إذا كان BX = 0003h , AX = fffch , DX = ffffh

4/ وضع محتويات المسجلين AH , AL fu] تنفيذ كل من الاوامر التالية:

أ/ DIV BL اذا كــان DL = 03h , AX = 000Dh

ب/ Idiv BL اذا كــان BL = Ffh , AX = FFFBh

ج/ Div BL اذا كان BL = 10h , AX = 00ffh

د/ Div BL اذا كــان BL = 02h , AX = FFE0h

5/ وضع محتويات المسجل DX بعد تنفيــذ الأمــر CWD إذا كان المسجل AX يحوى الأرقام التالية:

أ/ 7E02 ب/ 8ABCh ج/ 1ABCh

6/ وضع محتويات المسجل AX بعد تنفيذ الأمــر CBW إذا كان المسجل AL يحوى الأرقام التالية:

أ/ F0h ب/ 5Fh ج/ 80h

7/ أكتب جزء من برنامج بلغــة التجميــع بحيــث يقــوم بحســاب كــل مــن المعــادلات التاليــة باعتبــار أن المتغيرات C , B , A من النوع Word وانه لا يوجد فيضان

```
a-   A  =  5 x A - 7
b-   B  =  (A - B ) * (B - 10)
c-   A  =  6  - 9 * A
d-   if   A2  + B2  =  C2  then
          set  cf
     else
          clear cf
     end_if
```

البرامج

لاحظ أن بعض هذا البرامج تفترض استخدام الإجراءات Outdec , Indec والتي تم كتابتها في هذا الفصل.

8/ قم بتعديل الإجراء INDEC ليقوم بالتأكد من حدوث فيضان

9/ أكتب برنامج يقوم بسؤال المستخدم بإدخال الزمن بالثواني (حتى 65535) يقوم البرنامج بطباعة الزمن بالساعات والدقائق والثواني مع رسالة مناسبة.

10/ قم بكتابة برنامج يقوم بقراءة كسر على الصورة (M/N) حيث M < N يقوم البرنامج بطباعة

النتيجة في صورة كسر عشري وذلك حسب الخوارزمية التالية:

1. Print “.”
2. Divide 10 x M By N , getting Quotient Q & Remainder R
3. Print Q
4. Replace M By R & go to step 2

استخدام الإجراء INDEC لقراءة الرقمين N , M

11/ أكتب برنامج يقوم بحساب القاسم المشترك الأكبر (GCD) Greatest common Divisor صحيحين M , N وذلك حسب الخوارزمية التالية.

Divide M by N , getting Quotient (1) and remainder R

If R = 0 , stop N is the GCD of M and N

If R <> 0 , Replace M by N by R and Repeat step 1

الفصل التاسع

المصفوفات وطرق العنونة المختلفة

Arrays and addressing Modes

في بعض التطبيقات نحتاج لتجميع المعطيات في مجموعات فمثلاً قد نحتاج لقراءة درجات الطلاب في مادة محدده في هذه الحالة يمكن تعريف عدد من المتغيرات يساوى عدد الطلاب وفي هذه الحالة يصعب كتابة برنامج يقوم بالتعامل مع كل الطلاب ولهذا السبب نلجأ لتجميع هذه الدرجات في مصفوفة عدد عناصره هو عدد الطلاب. وبهذه الطريقة يمكن التعامل مع المصفوفة باستخدام الفهرسة وبالتالي يمكن جمع عناصر المصفوفة أو إيجاد المتوسط أو الانحراف المعياري يتم ذلك عن طريق مسح المصفوفة من أوله وإجراء العملية المطلوبة.

في هذا الفصل سنوضح كيفية تعريف المصفوفات المختلفة ثم نتعرض لأنماط العنونة المختلفة والتي سنحتاج لها لمخاطبة عناصر المصفوفة في البرنامج. ثم نتعرف على طريقة تعريف المصفوفة

المصفوفات ذات البعد الواحد One - Dimensional Arrays

المصفوف هو عبارة عن مجموعة من العناصر مرتبة وراء بعضها في الذاكرة وقد تكون هذه العناصر عبارة عن حروف Bytes أو جمل Words أو أي نوع آخر. فإذا كان اسم المصفوف هو A فأن عناصر المصفوف هي A[1] و A[2] و A[3] ....A[N] حيث N هو عدد عناصر المصفوف وقد تعرفنا سابقاً على كيفية تعريف المصفوف فمثلاً لتعريف مصفوف من الحروف اسمه Msg نستخدم التعريف

MSG DB "ABCDE"

حيث يتم يكون MSG[1] = A و (B) = MSG[2] وهكذا .

ولتعريف مصفوف من الكلمات (كل عنصر يشغل خانتين في الذاكرة) باسم A نستخدم التعريف التالي :

A DW 10,20,30,40,50,60

حيث يتضمن ذلك تعريف مصفوف به 5 خانات كل خانه عبارة عن كلمه Word بقيم ابتدائية A[1] = 10 و A[2] = 20 و A[3] = 30 و A[4] = 40 و A[5] = 50

يسمى عنوان المصفوف بالعنوان الأساسي للمصفوف Base Address of the array ويتم تحديد هذا العنوان عند تحميل البرنامج إلى الذاكرة فمثلاً إذا كان عنوان الإزاحة للمصفوف A هو العنوان 0200h يكون شكل المصفوف على النحو التالي:

العنوان الرمزي	قيمة الإزاحة	المحتويات في النظام العشري
A	0200h	10
A + 2h	0202h	20
A + 4h	0204h	30
A + 6h	0206h	40
A + 8h	0208h	50

المؤثر DUP (Duplicate)

يستخدم المؤثر Dup لتعريف مصفوف بعدد من العناصر تأخذ كلها نفس القيمة الابتدائية ويكون على الصورة.

Repeat_Count Dup (value)

يقوم المؤثر Dup بتكرار القيمة value عدد من المرات يساوى Repeat_count مثلاً:

GAMMA DW 100 Dup (0)

هنا يتم تعريف مصفوفة باسم GAMMA يحتوي على 100 عنصر كل عنصر عبارة عن Word ووضع قيمة ابتدائية 0 في كل العناصر وكمثال آخر.

DELTA DB 60 Dup (?)

حيث يتم تعريف مصفوفة باسم Delta يتكون من 60 عنصر حرفي Byte وعدم وضع أي قيمة ابتدائية للعناصر

ما هي محتويات الذاكرة عند العنوان line وذلك عند تعريفه على الصورة التالية:

Line DB 5 , 4 , 3 DUP (2 , 3 DUP (0) , 1) مثلاً التعريف التالي

Line DB 5 , 4 , 2,0,0,0,1,2,0,0,0,1,2,0,0,0,1 يطابق التعريف

مواقع عناصر المصفوفة

يبدأ تخزين المصفوفة في الذاكرة ابتدأ من العنوان الأساسي للمصفوفة وهو عنوان العنصر الأول ويكون عنوان العنصر الثاني يعتمد على نوعية عناصر المصفوفة فإذا كانت Byte يكون هو الأساسي + 1 أما إذا كانت Word يكون عنوان العنصر الثاني هو العنوان الأساسي + 2

وهكذا وعموماً إذا كانت S هي طول عنصر المصفوف(1= S إذا كانت العناصر عبارة عن Byte و2 = S إذا كانت العناصر عبارة عن Word) يكون عنوان العنصر N هو العنوان الأساسي للمصفوف + S * (N - 1) فمثلاً المصفوف A المعرف سابقاً يكون فيه عنوان العنصر N هو

A + (N - 1) S

مثال: استبدل العنصرين رقم 10 ورقم 25 في المصفوف W حيث (?) W DW 100 Dup

الحل

عنوان العنصر العاشر هو W + (10 - 1) * 2 =

W + 9 x 2 = W + 18

وعنوان العنصر 25 هو W + (25 - 1) * 2 = W

+ 24 x 2 = W + 48

وبالتالي يكون البرنامج هو

```
MOV     AX , W + 18
XCHC    A x , W + 48
MOV     W + 18 , A x
```

في كثير من التطبيقات نحتاج للتعامل مع عناصر المصفوف كلها. مثلاً إذا أردنا إيجاد مجموع

عناصر المصفوف A والذي به عدد N عنصر فإننا نحتاج لمخاطبة العناصر داخل حلقة كما في الخوارزمية التالية:

```
Sum = 0
M = 0
Repeat
    Sum = sum + A [M]
    M = M + 1
Until    M = N
```

ولعمل ذلك نحتاج لطريقة للتحرك بين عناصر المصفوف وذلك باستخدام مؤشر محدد وتغيير قيمته كل مره داخل الحلقة ولذلك سنقوم في الجزء التالي بتوضيح طرق العنونة المختلفة المستخدمة.

أنماط العنونة ADDRESSING MODES

طريقة استخدام معاملات الأمر تسمى بطرق العنونة وقد تعاملنا سابقاً مع ثلاثة أنماط مختلفة للعنونة وهى:

1/ نمط المسجلات Register Mode

وفيه يتم استخدام أحد المسجلات المعروفة مثل

MOV A x , B

2/ النمط اللحظي Immediate Mode

وفيه يتم استخدام الثوابت بمعاملات مثل

MOV A x , 5

هنا المعامل Ax يعتبر عنونه من النوع Register والمعامل 5 يعتبر من النمط اللحظي Immediate

3/ النمط المباشر Direct Mode

حينما يكون المعامل أحد المتغيرات مثل

Ax , Words MOV

حيث المعامل Words عبارة عن مجموعة مباشرة

هناك أربعة أنماط أخرى سنقوم بالتحدث عنها في الأجزاء التالية:

4/ نمط العنونة بالاستخدام الغير مباشر للمسجلات .Register Indirect Mode

يتم هنا تحديد عنوان الذاكرة المطلوب في أحد المسجلات SI أو BX أو DI أو BP وعلى هذا يعتبر المسجل أنه مؤشر Pointer للعنوان المطلوب مخاطبته ويتم وضع المعامل داخل الأمر على الصورة التالية:

[Register]

المسجلات DI , SI , BX تشير إلي العناوين داخل مقطع البيانات DS والمسجل BP يشير إلي العناوين داخل مقطع المكدس SS.

مثال:

إذا كان SI = 0100h والكلمة في العنوان 0100h في البيانات تحتوى على الرقم 1234h فإن الأمر

MOV AX , [SI]

يتم أخذ القيمة 100h من المسجل SI وتحديد العنوان DS: 0100 ثم أخذ القيمة الموجودة في ذلك العنوان (الرقم 1234h) ووضعها في المسجل AX (أي AX = 1234h) وهذا بالطبع غير الأمر

MOV AX , SI

والذي يقوم بوضع الرقم 0100h في المسجل AX

مثال:

افترض أن DI = 3000h , SI = 2000h , BX = 1000h وأن الذاكرة تحوى القيم التالية في مقطع البيانات في الازاحه 1000h يوجد الرقم 1BACH وفي الازاحه 2000h يوجد

الرقم 20FEh وفي الإزاحة 3000h يوجد الرقم 031Dh حيث أن الازاحات أعلاه في مقطع البيانات Data Segment . حدد أياً من الأوامر أدناه صحيحاً. ووضع العدد الذي يتم نقله في هذه الحالة:

أ - MOV BX, [BX]　　　ب - MOV CX, [SI]

جـ- MOV BX, [AX]　　　د - ADD [SI] [DI]

هـ - INC [DI]

الحل:

أ - MOV BX, [BX] يتم وضع الرقم 1BACh في المسجل BX

ب - MOV CX, [SI] يتم وضع الرقم 20FEh في المسجل CX

جـ MOV BX, [AX]– خطأ لا يمكن استخدام المسجل AX في العنونة الغير مباشرة.

د - ADD [DI], [SI] خطأ لا يمكن جمع محتويات عنصرين في الذاكرة بأمر واحد

هـ/ INC [DI] يتم جمع الرقم واحد إلى محتويات الذاكرة في الازاحه 3000h لتصبح القيمة 031Eh الموجودة

مثال: أكتب جزء من برنامج يقوم بجمع العناصر العشرة للمصفوفة W في المسجل AX إذا كان

W DW 10,20,30,40,50,60,70,80,90,100

الحل:

يتم استخدام المسجل SI كمؤشر ووضع القيمة صفر فيه وبعد ذلك في داخل حلقة يتم قراءة العنصر ثم جمع الرقم 2 (لأن عناصر المصفوفة عبارة عن كلمات Word) إلي المسجل SI كما يلي:

```
        XOR     AX , AX
        LEA     SI , W
        MOV     CX , 10
ADDNOS :
        ADD     AX , [SI]
        ADD     SI ,2
        LOOP    ADDNOS
```

مثال: أكتب إجراء يسمى REVERSE والذي يقوم بعكس مصفوفة مكون من N عنصر كلمات Words) وذلك بتعديل العنصر الأول مع الأخير والثاني مع العنصر السابق للأخير وهكذا).

الحل: إذا كان N هو عدد عناصر المصفوفة يتم تكرار الحلقة N/2 مره وفي كل مره يتم استبدال عنصرين أحدهما يشير إليه المسجل S1 والثاني يشير إليه المسجل D1 ولعمل ذلك يجب جعل المسجل SI يشير إلى أول عنصر في المصفوفة والمسجل DI يشير إلى آخر عنصر. داخل الحلقة يتم عمل تجهيز المسجلين SI , DI وذلك بجمع الرقم 2 إلى المسجل SI وطرح الرقم 2 من المسجل DI (وذلك لأن عناصر المصفوفة هي كلمات Words).

```
REVERSE        PROC
;      عكس عناصر مصفوفة
; Inputs :    SI  يشير الى عنوان الازاحة للمصفوفة
;          BX عدد عناصر المصفوفة
; Outputs : SI يشير إلي المصفوفة بعد عكسه
     Push        AX
     Push    BX
     Push        CX
     Push        SI
     Push        DI
```

```
; D1 يشير الى آخر عنصر
    Mov        DI  , SI
    Mov        Cx  , Bx      ;    Cx   =
n
    Dec        BX      ;    Bx  =  n  -
S
    SHL    BX , 1
    ADD        DI , Bx  ;    DI  =  SI +
2 (n - 1)
    ShR        Cx , 1   ;    Cx = n/2
XCHG_Loop:
    Mov        AX  , [SI]
    XCHC   AX , [DI]
    Mov        [SI],  AX
    ADD        SI ,2
    Sub        DI , 2
    Loop   XCHg_Loop
    Pop        DI
    Pop    SI
    Pop        CX
    Pop        BX
    Pop        AX
    RET
REVERSE   ENDP
```

5/ أنماط العنونه المفهرسة والأساسية Indexed and Based Addressing modes

في هذه الأنماط يتم إضافة عدد يسمى بالازاحة Displacement لمحتويات المسجل وقد تكون الازاحه أحد القيم التالية حيث A عبارة عن متغير تم تعريفه.

- قيمة الازاحه لمتغير مثل A
- قيمة ثابتة مثل 2
- قيمة الازاحه لمتغير بالاضافه الى قيمة ثابتة باشارة مثل 2 + A ويأخذ هذا النمط إحدى الصور التالية :

[Register +Displacement]
[Displacement + Register]
[Register] +Displacement
Displacement + [Register]
Displacement [Register]

المسجل يجب أن يكون أحد المسجلات BX و BP و SI و DI إذا تم استخدام أحد المسجلات BX أو SI أو DI فإن المسجل DS يشير إلي المقطع المعني أما إذا تم استخدام المسجل BP فإن المسجل SS يشير إلي المقطع المعني.

إذا تم استخدام المسجل BX أو المسجل BP يسمي النمط بـ Based بينما يسمي النمط بـ

Indexed إذا تم استخدام المسجل SI أو المسجل DI.

كمثال لهذا النمط إذا كان المتغير W عبارة عن مصفوفة من الجمل Word Array وأن المسجل BX به الرقم 4 فإن الأمر التالي يقوم بوضع العنصر الموجود في الذاكرة بالعنوان 4 + W في المسجل AX

```
MOV      AX ,  W [ BX]
```

وهذا هو العنصر الثالث في المصفوفة، ويمكن كتابة الأمر بأحد الصور التالية والتي تؤدي نفس الغرض:

```
MOV      AX , [ W + BX ]
MOV      AX , [ BX + W ]
MOV      AX , W +[ BX ]
MOV      AX , [ BX ] + w
```

كمثال آخر افترض أن المسجل SI يحتوي على عنوان بداية مصفوفة W من الجمل Word Array.

أي من الأوامر التالية يقوم بوضع محتويات العنصر الثاني والموجود بالعنوان 2 + W في المسجل AX :

```
MOV      AX , [ SI + 2 ]
MOV      AX , [ 2 + SI ]
```

```
MOV     AX , 2 + [ SI ]
MOV     AX , [ SI ] + 2
MOV     AX , 2 [ SI ]
```

مثال

أكتب (مستعملاً نم العنونة الأساسية) جزء من برنامج يقوم بجمع عناصر المصفوف W في المسجل AX إذا كان:

W DW 10,20,30,40,50,60,70,80,90,100

الحل:

```
        XOR     AX , AX
        XOR     BX , BX
        MOV     CX , 10
ADDNOS:
        ADD     AX , w [ BX ]
        ADD     BX , 2
        LOOP    ADDNOS
```

يتم إضافة الرقم 2 للمسجل SI للتحرك للعنصر التالي حيث أن المصفوف به كلمات Words

مثال

افترض أن المتغير Alpha معرف علي النحو التالي :

ALPHA DW 0123h, 0456h, 0789h, 0abcdh

وأن المسجلات بها القيم التالية : SI =4, BX = 2

DI = 1 وأن الذاكرة بها الرقم 1084h في الإزاحة 0002 وبها الرقم 2BACh في الإزاحة 0004.

وضح أياً من الأوامر التالية صحيح وإذا كان الأمر صحيح وضح عنوان الإزاحة للمصدر والرقم الذي تم التعامل معه في كل من الحالات التالية:

```
a.  MOV   AX , [ALPHA + BX ]
b.  MOV   BX , [ BX+ 2 ]
c.  MOV   CX , ALPHA [ SI ]
d.  MOV   AX , -2 [ SI ]
e.  MOV   BX , [ALPHA + 3 + DI ]
f.  MOV   AX , [ BX ] 2
g.  ADD   BX , [ALPHA + AX ]
```

الحل:

القيمة التي تم وضعها في المسجل	عنوان الإزاحة	السؤال
0456h	APLPHA +2	A
2BACh	2 + 2 = 4	B
0789h	ALPHA + 4	C
1084h	-2 + 4 = 2	D

0789h	ALPHA + 3 + 1	E
	المصــــدر مكتــــوب بطريقة غير صحيحة	F
	لا يمكـــن اســـتخدام المسجل AX هنا	G

المعامل PTR والإيعاذ LABEL:

ذكرنا فيما سـبق أن المعـاملين للأمـر يجـب أن يكونا من نفس النوع فمثلاً يكون المعـاملان مـن النوع الحرفي Byte أو من النـوع WORD . وإذا كان المعامل عبارة عن رقم ثابـت يقـوم المجمـع بتفسيره حسب نوع المعامل الثاني فمثلاً يتم التعامل مع الرقم الثابت في المثال التالي على أنه عبـارة عن متغير من النوع WORD .

```
MOV     AX , 1
```

بينما يتم التعامل مع الثابت التالي على انه متغير حرفي Byte

```
MOV     AL , 1
```

ولكن لا يمكن التعامل مع الأمر التالي

```
MOV     [ BX ] ,  1
```

وذلك لأن المستودع غير معرف هل هو word أم Byte .

ليتم تخزين الثابت على أنه من النوع Byte نستخدم الأمر

```
MOV    BYTE    PTR   [ BX ] , 1
```

وليتم تخزين الثابت على أنه من النوع WORD نستخدم الأمر

```
MOV    WORD    PTR   [ BX ] , 1
```

مثال: استبدل الحرف الأول في متغير يسمي MSG بالحرف "T"

الحل:

الطريقة الأولى :

باستخدام طريقة العنونة الغير مباشرة باستخدام المسجلات Register indirect mode

```
LEA     SI, Msg
MOV     BYTE    PTR    [SI],    'T'
```

الطريقة الثانية: باستخدام العنونة المفهرسة Index Mod

```
XOR     SI, SI
```

MOV mSG[SI] , 'T'

غير ضروري هنا استخدام المعامل PTR حيث أن Msg عبارة عن متغير حرفي

استخدام PTR لإعادة تعريف متغير:

يمكن استخدام PTR لإعادة تعريف متغير تم تعريفه من قبل والصيغة العامة هي:

Type PTR Address_Expression

حيث Type هي Byte أو WORD أو Dword و Address_Expression هى DB أو DW أو DD

فمثلاً إذا كان لدينا التعريف التالي:

```
DOLLARS   DB    1Ah
CENTS     DB    52h
```

إذا أردنا وضع محتويات المتغير Dollars في المسجل AL والمتغير Cents في المسجل AH باستخدام أمر واحد لن نستطيع ذلك

```
MOV   AX  , DOLLARS    ; ILLEGAL
```

حيث أن المصدر عبارة عن Byte بينما المستودع عبارة Word ولكن يمكن إعادة كتابة الأمر على الصورة التالية

```
MOV    AX ,word   PTR  DOLLARS   ;
AL=DOLLARS , AH =Cents
```

وسيتم وضع الرقم 521Ah في المسجل AX

المعامل LABEL:

يمكن حل مشكلة اختلاف الأنواع هـذه باسـتخدام المعامل LABEL فمـثلاً يمكـن اسـتخدام الإعـلان التالي:

MONEY	LABEL	WORD
DOLLARS	DB	1Ah
CENTS	DB	52h

وبالتالي يستخدم المتغير MONEY على انـه مـن النوع Word والمتغيـرين DOLLARS و CENTS عبارة عن متغيرات من النـوع Byte . وبالتـالي يصبح الأمر التالي صحيحاً

```
MOV        Ax , Money
```

وله نفس تأثير الأمرين

```
MOV    AL , DOLLARS
MOV    AH , CENTS
```

مثال: اعتبر الإعلانات التالية:

```
.DATA
    A    DW      1234H
```

```
B    LABEL      BYTE
     DW     5678H
C    LABEL      WORD
C1   DB     9AH
C2   DB     0BCH
```

تكون الأوامر على النحو التالي:

الرقم	الأمر	ملحوظة	البيانات المنقولة
1	MOV AX , B	غير صحيح	تضارب الأنواع
2	MOV AH , B	صحيح	78h
3	MOV CX , C	صحيح	0BC9Ah
4	MOV BX , WORD PTR B	صحيح	5678h
5	MOV DL , BYTE PTR C	صحيح	9Ah
6	MOV AX , WORD PTR C1	صحيح	0BC9AH

تجاوز المقطع Segment Override

في نمط العنونة الغير مباشر باستخدام المسجلات Registers تستخدم المسجلات BX و SI و DI للعنونة في داخل مقطع البيانات DS. يمكن استخدام

هذه المسجلات لتحديد عناوين في مقطع آخر وذلك على النحو التالي:

[Segment_Register : Pointer_Register]

مثلاً الأمر

MOV Ax , ES : [SI]

يؤدى لنقل البيانات في الذاكرة في المقطع ES والإزاحة SI إلي المسجل AX وتستمر هذه الطريقة في مخاطبة بيانات في أكثر من مقطع في نفس الوقت مثل نقل البيانات من مكان لآخر بعيد في الذاكرة.

الوصول إلي المكدس Accessing the Stack:

ذكرنا أن المسجل BP يستخدم مع مسجل المقطع SS وذلك للتخاطب مع مقطع المكدس وبالتالي يمكن قراءة بيانات المكدس.

مثال:

أنقل محتويات أعلى ثلاث خانات في المكدس في المسجلات AX , B X , CX المكدس وذلك دون تغيير محتويات المكدس.

الحل:

```
MOV    BP , SP
MOV    AX , [ BP]
MOV    BX , [ BP + 2]
MOV    CX , [ BP + 4]
```

تطبيق: ترتيب مصفوفة:

هنالك طرق عديدة لترتيب محتويات مصفوفة .

ونتناول هنا إحدى هذه الطرق وهي طريقة الترتيب بالاختيار Select Sort

لترتيب مصفوفة به N عنصر يتم ذلك على النحو التالي

المرة الأولي: أوجد العنصر الأكبر في العناصر من A [1] إلي A [N] وقم باستبداله مع العنصر A [N] وبالتالي ستحتاج لترتيب العناصر من 1 إلي N - 1

المرة الثانية: أوجد العنصر الأكبر في العناصر من A [1] إلي A [N - 1] وقم باستبداله مع العنصر A [N-1] وبالتالي ستحتاج لترتيب العناصر من 1 إلي N - 2

المرة N - 1: أوجد العنصر الأكبر في العناصر من A [I] إلي A [2] وقم باستبداله مع العنصر A [1]
وبهذا تكون عملية الترتيب قد اكتملت

وسنتابع الجدول التالي عليه الترتيب:

الموقع	1	2	3	4	5
البيانات ألأوليه	21	5	16	40	7
المـــــرة الأولى	21	5	16	7	40
المـــــرة الثانية	7	5	16	21	40
المـــــرة الثالثة	7	5	16	21	40
المـــــرة الرابعة	5	7	16	21	40

وتكون الخوارزمية على النحو التالي:

i = N
For N - 1 Times Do
Find the position K of the Largest element among A [1] .. A [I]

```
        SWAP   A [K]  and   A [ 1 ]
        I  :  =  I  -  1
    End_For
```

بلغة التجميع :

```
SELECT  PROC
    ;SORTS A BYTE ARRAY BY THE
SELECTSORT METHOD
    ;INPUTS:SI= ARRAY OFFSET ADDRESS
    ;       BX=NUMBER OF ELEMENTS
    ;OUTPUTS:SI=OFFSET OF SORTED ARRAY
    ;USES:SWAP
    PUSH    BX
    PUSH    CX
    PUSH    DX
    PUSH    SI

    DEC     BX
    JE      END_SORT
    MOV     DX , SI
SORT_LOOP:
    MOV     SI , DX
    MOV     CX , BX
    MOV     DI , SI
    MOV     AL , [DI]
```

```
FIND_BIG:
      INC     SI
      CMP     [SI], AL
      JNG     NEXT
      MOV     DI , SI
      MOV     AL , [DI]
NEXT:
      LOOP    FIND_BIG
      CALL    SWAP
      DEC     BX
      JNE     SORT_LOOP
END_SORT:
      PUSH    SI
      PUSH    DX
      PUSH    CX
      PUSH    BX
SELECT  ENDP

SWAP    PROC
      ;INPUT: SI=ONE ELEMENT
      ;       DI=OTHER ELEMENT
      ;OUTPUT:EXCHANGED ELEMENTS
      PUSH    AX
      MOV     AL , [SI]
      XCHG    AL , [DI]
      MOV     [SI] , AL
      POP     AX
```

```
        RET
SWAP    ENDP
```

يستقبل الإجراء SELECT السابق عنوان ألازاحه لبداية المصفوف في المسجل SI وعدد عناصر المصفوف N في المسجل BX .

ويمكن تجربه البرنامج باستخدام البيانات التالية مع البرنامج الموضح لترتيب عناصر المصفوف A

```
TITLE SORT: SELECT SORT PROGRAM
.MODEL SMALL
.STACK 100H
.DATA
        A    DB  5 , 2 , 1 ,  3 , 4
.CODE
MAIN PROC
        MOV     AX , @DATA
        MOV     DS , AX
        LEA     SI , A
        CALL    SELECT
        ;dos exit
        MOV AH,4CH
        INT 21H
MAIN    ENDP
        INCLUDE  PROCFILE.ASM
```

END MAIN

ويمكن تجربة البرنامج باستخدام برنامج Debug على النحو التالي : حيث يتم تشغيل البرنامج إلي عنوان بداية الإجراء علي النحو التالي

```
-GC
AX=100D  BX=0005 CX=0049 DX=0000 SP=0100
Bp=0000 SI=0004 DI=0000
DS=100D ES=0FF9 SS=100E CS=1009 IP=000C
NV UP EI PL NZ NA PO NC
1009:000C  E80400 CALL  0013
```

قبل نداء الإجراء يتم استعراض محتويات المصفوف

```
-D 4 8
100D:0000  05 02 01 03- 04
```

والآن يتم استدعاء الإجراء

```
-GF
AX=1002  BX=0005 CX=0049 DX=0000 SP=0100
Bp=0000 SI=0004 DI=0005
DS=100D ES=0FF9 SS=100E CS=1009 IP=000F
```

```
NV UP EI PL ZR NA PE NC
1009:000F  B44C MOV  AH , 4C
```

والآن يتم استعراض محتويات المصفوف بعد ترتيبه

```
-D 4 8
100D:0000  01 02 03 04- 05
```

المصفوف ذو البعدين:

المصفوف ذو البعدين عبارة عن مصفوف يتم التخاطب مع كل عنصر بتحديد رقم الصف ورقم العدد حيث يكون العنصر B [I, 1] هو العنصر الذي يقع رقم 1 والعدد رقم 8

كيفية تخزين المصفوف:

لان الذاكرة عبارة من مصفوف عبارة عن صف واحد يجب تخزين عناصر المصفوف بصوره تسلسليه وعلى ذلك توجد طريقتين لتخزين المصفوف ذو البعدين

1. صف_صف Row Major Order

حيث يتم تخزين الصف الأول كله مصفوفاً الصف الثاني وهكذا

2. عمود_عمود Column Major Order

حيث يتم تخزين العمود الأول كله متبوعاً بالعمود الثاني وهكذا

وكمثال لذلك كان لدينا مصفوف B به 3 صفوف و 4 أعمدة وبه العناصر 10 و 20 و 30 و 40 في الصف الأول و 50, 60, 70, 80 في الصف الثاني و 90 , 100 ، 110 , 120 في الصف الثالث.

قد يتم تخزين المصفوف في صورة صف_صف على النحو التالي

```
B    DW       10, 20,30,40

     DW       50 , 60 , 70 , 80
     DW       90 , 100, 110 , 120
```

ويمكن تخزينه في صورة عمود-عمود على النحو التالي:

```
B         DW       10 , 50 , 90
     DW       20 , 60 , 100
     DW       30 ,  60 , 110
     DW 40  ,  80  , 120
```

أكثر لغات البرمجة العليا تقوم بتعريف المصفوف في صورة صف_صف . وفي لغة التجميع يمكن التعامل مع أي الطريقتين بدون مشاكل حيث نفضل

طريقة صف_صف إذا كانت عناصر الصف الواحد يتم التعامل بها في حلقة محدده كما نفضل طريقة عمود_عمود إذا كان التعامل مع العمود كله يتم في حلقة محدده .

وكما لاشك انه عند التعامل مع المصفوف في إحدى اللغات العليا وإعادة التعامل معه بلغة أخرى يجب اعتبار طريقة تخزين المصفوف في اللغتين وإلا ستحدث أخطاء عديدة إذا تم تخزين المصفوف في صورة صف_صف وتم قراءته على صورة عمود_عمود

<u>تحديد عنوان العنصر:</u>

افترض أن المصفوف A به M صف و N عمود وانه قد تم تخزينه في صورة صف_صف وأن S هو عدد الخانات المطلوبة لتخزين عنصر واحد هو (لاحظ أن S=1 في حالة تخزين عناصر عبارة عن Byte و S=2 في حالة تخزين عناصر عبارة عن Word) . المطلوب تحديد عنوان العنصر A [I , j]؟

سنقوم بتحديد العنوان على طريقتين:

1. إيجاد مكان أول عنصر في الصف رقم I

2. إيجاد مكان العنصر رقم j في ذلك الصف

العنصر في الصف الأول يتم تخزينه في العنوان A

ولان عدد العناصر في كل صف هو N عنصر

العنصر الأول في الصف الثاني يتم تخزينه في العنوان A + s * N

العنصر الأول في الصف الثالث يتم تخزينه في العنوان A + 2 * N * S

العنصر الأول في الصف I يتم تخزينه في العنوان A + (I - 1) N * S

الآن الخطوة الثانية:

العنصر رقم j سيتم تخزينه في مكان يبعد (j - 1) * s من عنوان بداية الصف المحدد (حيث (j - 1) هو عدد العناصر السابقة لهذا العنصر في الصف) وعلى ذلك يصبح عنوان العنصر A [I , j] في المصفوف المخزن على صورة صف_صف هو

$$A + (i - 1) \times N \times s + (j - 1) \times s$$

وإذا تم تخزين المصفوف في صورة عمود_عمود نفس الطريقة السابقة سنجد أن عنوان العنصر A [I , j] هو

$$A + (j-1) \times M \times S + (I-1) \times s$$

مثال:

المصفوف A يحتوى على M صف و N عمود مخزن في صورة صف_صف

1. أذكر عنوان بداية الصف رقم I
2. أذكر عنوان بداية العمود رقم j
3. كم خانة تقع بين عنصرين في نفس العمود

الحل

1. بالتطبيق في القانون نجد أن عنوان بداية الصف رقم I هو

$$A + (I-1) + N \times S$$

2. بالتطبيق في القانون نجد أن عنوان بداية العمود رقم j هو

$$A + (j-1) \times S$$

3. لان لدينا من عنصر في صف فان عدد الخانات بين عنصرين متجاورين في عمود هي $N \times S$

نمط العنونة القاعدي المفهرس - based indexed:

في هذا النمط يكون عنوان الإزاحة للمعامل هو عبارة عن مجموع

1. محتويات مسجل القاعدة (BX أو BP)
2. محتويات مسجل الفهرسة (SI أو DI)
3. اختيارياً مسجل عنوان الإزاحة لمتغير
4. اختيارياً عنوان ثابت الإزاحة (موجب أو سالب)

إذا تم استخدام المسجل BX يكون ذلك في المقطع المحدد بالمسجل DS

إذا تم استخدام المسجل BP يكون ذلك في المقطع المحدد بالمسجل SS

ويتم كتابة المعامل بأكثر من طريقة مثل

1. Variable [Base_Register] [index_Reg]
2. [Base_Reg + index_Reg + VAR + const]
3. VAR [Base_Reg + index_Reg + Const]
4. Const [Base_Reg + Index + Var]

وترتيب العناصر عند كتابة المعامل اختيارياً

مثلاً افترض أن W متغير كلمة فإذا كانت محتويات المسجل BX هي الرقم 2 وان المسجل SI يحتوى

على الرقم 4. الأمر التالي بصوره المختلفة يقوم بوضع محتويات الذاكرة عند العنوان W+6 في المسجل Ax

MOV AX , W [BX] [SI]
MOV AX , W [BX+ SI]
MOV AX , [W + BX + SI]
MOV AX , [BX + SI] W

ويتم استخدام هذا النمط عادة عند التعامل مع المصفوفات ذات البعدين

مثال: مصفوفة A به 5 صفوف و 7 أعمده به عناصر عبارة عن words مخزن في صورة صف_صف اكتب مستخدماً نمط العنونة Based - Indexed جزء من برنامج يقوم بالآتي:

1. وضع الرقم 0 في عناصر الصف الثالث
2. وضع الرقم 0 في عناصر العمود الرابع

الحل: 1- أول عنصر في الصف الثالث يقع في العنوان

A + (3 - 1) x 7 x 2 = A + 2 x 7 x 2 = A + 28

MOV Bx, 28
XOR S I ,SI

```
              MOV     Cx , 7
CLEAR :       MOV     A [ Bx] [SI] , 0
              ADD     SI  , 2
              LOOP    CLEAR
```

2- أول عنصر في العمود الرابع يقع في العنوان

A + (4 - 1) x 2 = A + 3 x 2 = A + 6

يوجد عدد 14 عنصر (7 x 2) بين كل عنصرين متجاورين في العمود الواحد

```
              MOV     SI , 6
              XOR     BX ,BX
              MOV     Cx , 5
CLEAR :       MOV     A [ Bx] [SI] , 0
              ADD     BX  , 14
              LOOP    CLEAR
```

الأمر XLAT :

في بعض التطبيقات نحتاج لتحويل البيانات من صورة لأخرى. يتم استخدام الأمر XLAT (وهو بدون معاملات) لتحويل Byte بأخرى محددة في جدول حيث يتم تحويل محتويات المسجل AL ويحتوى المسجل BX على عنوان الإزاحة لبداية الجدول ويقوم الأمر بالآتي :

1. جمع محتويات المسجل AL إلي المسجل BX لتحديد عنوان العنصر المطلوب
2. وضع محتويات الذاكرة عند ذلك العنوان في المسجل AL

مثلاً:

افترض أن المسجل AL به رقم يقع بين 0h و Fh ونريد استبداله بالكود ASCII المناظر (مثلاً يتم استبدال 6h بـ 36h و0ch بـ 42h أى 'B' .....)

```
TABLE  DB 30h, 31h,32h, 33h, 34,35h,
36h, 37h, 38h, 39h
       DB 41h, 42h , 43h, 44h, 45h,
46h
```

وبعد ذلك يتم استخدام الأمر (مثلاً عند تحويل الرقم ch إلي الرقم 'c')

```
Mov     AL , och
LEA     BX, TABLE
XLAT
```

مثال:

البرنامج الموضح يقوم بتشفير رسالة محدده

(استبدال الحرف بحرف آخـــر مـــن جـــدول) وطباعة الرسالة مشفرة . ثم استعادة الرســالة مرة أخرى (باستخدام جــدول آخـــر) وطباعـــة الرسالة بعد استرجاعها.

```
TITLE secret message
.MODEL SMALL
.STACK 100H
.DATA
    CODE_KEY    DB 65 DUP('
'),'XQPOGHZBCADEIJUVFMNKLRSTWY'
                DB 37 DUP (' ')
    DECODE_KEY  DB 65 DUP('
'),'JHIKLQEFMNTURSDCBVWXOPYAZG'
                DB 37 DUP (' ')
    CODED       DB 80 DUP ('$')
    PROMPT      DB 'ENTER A MESSAGE :' , 0DH ,
0AH , '$'
    CRLF        DB 0DH , 0AH , '$'

.CODE

MAIN PROC
        ; initialize DS
        MOV     AX,@DATA
        MOV     DS,AX
        ;print user prompt
```

```
    LEA   DX,PROMPT
    MOV   AH,09H
    INT   21H
    ;READ AND ENCODE MESSAGE
    MOV   AH , 1
    LEA   BX , CODE_KEY
    LEA   DI , CODED
WHILE_:
    INT   21H
    CMP   AL , 0DH
    JE    END_WHILE
    XLAT
    MOV   [DI],AL
    INC   DI
    JMP   WHILE_
END_WHILE:
    ;GOTO NEW LINE
    MOV   AH , 9
    LEA   DX , CRLF
    INT   21H
    ;PRINT ENCODED MESSAGE
    LEA   DX,CODED
    INT   21H
    ;GOTO NEW LINE
    LEA   DX,CRLF
    INT   21H
    ;DCODE MESSAGE AND PRINT IT
```

```
        MOV     AH , 2
        LEA     BX , DECODE_KEY
        LEA     SI , CODED
WHILE2:
        MOV     AL , [SI]
        CMP     AL ,'$'
        JE      END_WHILE2
        XLAT
        MOV     DL ,AL
        INT     21H
        INC     SI
        JMP     WHILE2
END_WHILE2:
        ;return to DOS
        MOV AH,4CH
        INT 21H
MAIN    ENDP
        END MAIN
```

تمارين:

1. افترض الآتي:

المسجل AX يحتوى على الرقم 0500h

المسجل BX يحتوى على الرقم 1000h

المسجل SI يحتوى على الرقم 1500h

المسجل DI يحتوى على الرقم 2000h

الذاكرة عند العنوان 1000h تحتوى على الرقم 0100h

الذاكرة عند العنوان 1500 تحتوى على الرقم 0150h

الذاكرة عند العنوان 2000 تحتوى على الرقم 0200h

الذاكرة عند العنوان 3000 تحتوى على الرقم 0400h

الذاكرة عن العنوان 4000 تحتوى على الرقم 3000h

المتغير Beta متغير Word موجود عند الإزاحة 1000h

وضع عنوان الإزاحة للمصدر والقيمة التي يتم تخزينها في كل من الأوامر التالية (أن كانت صحيحة)

a- MOV DI , [SI] b- MOV DI , [DI]
c- ADD AX , [SI] d- SUB BX, , [DI]
e- LEA BX ,Beta [BX] f- ADD, SI], [DI]

g- ADD BH , [BL] h- ADD, AH, [SI]
c- MOV AX, [BX + DI + beta]

2. إذا أعطينا التعريف التالي

A DW 1,2,3
B DB 4,5,6
C LABEL word
Msg DB 'ABC'

افترض أن المسجل BX يحتوى على الإزاحة للمتغير C . أي من الأوامر التالية صحيح ووضح القيمة التي يتم وضعها في المسجل المستودع

a- MOV AH , BYTE PTR A
b- MOV AX , word PTR B
c- MOV AX , C
d- MOV AX , Msg
e- MOV AH , BYTE PTR C

3. استخدم المسجل BP للقيام بالآتي (لا تستخدم الأوامر push و pop)

أ/ استبدل قيمة أول جملتين في المكدس بصفر

ب/ انسخ أول 5 جمل في المكدس إلي المتغير ST_ARR بحيث يتم وضع الجملة الموجود في قمة المكدس في العنوان ST_ARR والكلمة التالية في العنوان ST_ARR+2 وهكذا

4. لدينا مصفوفين إحداهما A يحتوى على 10 عناصر من النوع word و B يحتوى على عنصر من النوع Byte

أ/ ضع في كل عنصر من المصفوف العنصر التالي له مباشرة (أي A [I] نضع فيها A[1 + 1] وهكذا) لكل العناصر وضع في العنصر الأخير A [10] العنصر الأول A[1].

ب/ ضع في المسجل DX عدد العناصر التى تحتوى على الرقم θ في المصفوف A .

جـ / افترض أن المصفوف B به رسالة. ضع في المسجل SI مؤشر للحرف 'E' إن وجد في الرسالة. إن لم يوجد في الرسالة الحرف 'E' ضع الرقم 1 في بيرق المحمول cf

5. أكتب إجراء يسمى Find_ij والذي يقوم بإرجاع عنوان الإزاحة للعنصر رقم I , J والموجود في الصف رقم I والعمود رقم j في مصفوف من الجمل مخزن في صورة صف_صف يقوم الإجراء باستقبال المتغير I في المسجل AX والمتغير i في المسجل BX وعدد الأعمدة N في المسجل Cx وعنوان الإزاحة لبداية

المصفوف في المسجل DX . يقوم المصفوف بإرجاع عنوان الازاحة للعنصر في المتغير DX .

برامج للكتابة:

6. المطلوب كتابة إجراء يسمى BUBBLE الذي يقوم باستقبال وترتيب مصفوف من الحروف وذلك باستخدام خوارزمية الترتيب المعروفة باسم Bubble Sort يقوم الإجراء باستقبال عنوان الإزاحة للمصفوف في المسجل SI وعدد العناصر في المسجل BX .

أكتب برنامج يقوم بسؤال المستخدم لإدخال سلسلة من الأرقام والمحتوية على خانه واحد فقط بينهما فراغ BLANK واحد فقط . قم بنداء الإجراء Bubble بعد ذلك قم بطباعة عناصر المصفوف والتي تم ترتيبها.

مثال للتنفيذ:

? 1 2 6 5 3 7
1 2 3 5 6 7

ملحوظة: تعمل الخوارزمية Bubble على النحو التالي

المرة الأولى: للعناصر J من 2 إلي N استبدل A [J] مع A [j - 1] إذا كان A [J] < A[J – 1]

سيتم بهذه العملية وضع أكبر عنصر في المكان رقم N

المرة الثانية: للعناصر J من 2 إلي N-1 استبدل A [J] مع A [j - 1] إذا كان A [J] < A[J – 1]

سيتم بهذه العملية وضع أكبر عنصر في المكان رقم N-1

:

المرة N - 1: إذا كان A[2] < A [1] استبدل العنصرين A [2] و A [1]

7. افترض التعريف التالي:

```
CLASS
        DB  'Ali            ' , 67 , 54 , 9 ,8 , 31
        DB  'HASSAN    ' , 30 , 50 , 59 ,42 , 53
        DB  'AHMED     ' , 65 , 73 , 85 ,18 , 90
```

حيث يتم تخزين الأسماء في 7 حروف

أكتب برنامج يقوم بطباعة اسم الطالب ومتوسط الدرجات التي أحرزها في الامتحانات مقرباً لعدد صحيح

8. أكتب برنامج يتعامل مع مصفوفة بـه 100 عنصـر بها قيم غير معرفة في البداية يقوم البرنامج بسؤال المستخدم لإدخـال حـروف (حـرف_حـرف) يقـوم البرنامج بعد قراءة كل حرف بترتيـب المصـفوفة وطباعته مرتباً. وبعد ذلك يقوم بسـؤال المسـتخدم البرنامج عند الضغط على مفتاح ESC.

مثال للتنفيذ:

```
?A
A
?D
AD
?B
ABD
?a
ABDa
?<esc>
```

9. أكتب إجراء يسمى PRINTHEX والذي يستخدم الأمر XLAT لطباعة محتويات المسـجل BX فـي الصورة السداسـية عشـر. جـرب الإجـراء بسـؤال المستخدم لإدخال رقم سداسي عشر مكـون مـن 4 خانات وذلك باستخدام الإجـراء IN_HEX والـذي قمت بكتابته في الأجزاء السابقة . ثم قـم بنـداء

الإجراء PRINTHEX لطباعة الرقم الذي تم إدخاله في بداية البرنامج.

الفصل العاشر

أوامر التعامل مع السلاسل String Instructions

في هذا الجزء سنتناول الأوامر التي نتعامل مع النصوص. وكما نعلم فإننا نتعامل مع النص على انه مصفوفة من الحروف وبالتالي لدينا مجموعة من الأوامر التي نتعامل مع هذه المصفوفات الخاصة فمثلاً لدينا أوامر للقيام بالتالي

* نسخ رسالة أو نص من مكان لمكان
* البحث عن حرف معين أو كلمة في سلسلة
* تخزين أحرف في سلسلة
* مقارنة سلسلة من الرموز أبجدياً

جميع هذه العمليات يمكن تنفيذها بمجموعة من الأوامر التي تستخدم أنماط العنونة المختلفة الموضحة في الجزء السابق ولكن هذه العملية تتطلب كتابة مجموعة من الأوامر وفي حالة استخدام أوامر خاصة بالنصوص يمكن أن يتم تنفيذها هنا

بأمر واحد فقط مما يجعل استخدام أوامر النصوص والرسائل اسهل.

بيرق الاتجاه DF:

بيرق الاتجاه هو أحد بيارق التحكم Control Flags وهو يحدد الاتجاه الذي سيتم فيه التعامل مع أوامر النصوص حيث يتم استخدام المسجلات DI , SI عند التعامل مع النصوص. وهناك طريقتان للتعامل مع النص. إما التعامل معه من البداية وفي هذه الحالة نجعل المسجل DI أو SI يشير إلي أول حرف في النص وبالتالي فان التعامل يتم بزيادة محتويات المسجلات لتشير إلى الحرف التالي وفي هذه الحالة يتم وضع الرقم 0 في البيرق DF.

وإذا تم وضع الرقم 1 في البيرق بمعنى ذلك أن التعامل مع النص يتم عند النهاية ويتم إنقاص محتويات مسجلات الفهرسة.

يتم وضع الرقم صفر في بيرق الاتجاه باستخدام الأمر

```
CLD ; clear Direction flag
```

ويتم وضع الرقم 1 في البيرق باستخدام الأمر

```
STD ; set Direction flag
```

ولا تؤثر هذه الأوامر في البيارق الأخرى.

نقل سلسلة Moving String:

إذا كان لدينا التعريف التالي:

```
String1   DB        'Hello'
String2         DB      5 Dup  ( ? )
```

وأردنا عمل نسخة من النص الأول في النص التالي وهذا يحدث عادة عندما نريد نسخه من رسالة أو عند دمج رسالتين في البرنامج .

يستخدم الأمر MOVSB وهو أمر بدون معاملات .
يستخدم الأمر لنقل محتويات الذاكرة في العنوان DS:SI إلى الذاكرة في العنوان ES:DI ولا يتم تغيير محتويات المصدر. بعد نقل الحرف يتم أوتوماتيكيا زيادة محتويات المسجلين DI:SI بواحد إذا كان بيرق الاتجاه يحتوى على الرقم 0 .
وكمثال على ذلك يمكن نسخ سلسلة(1) في المثال على سلسلة(2) بتنفيذ التالي:

```
MOV     AX , @DATA
MOV     DS, AX
MOV     ES, AX
LEA     SI , String1
LEA     DI , String2
```

```
CLD
MOVSB
MOVSB
:
```

يعتبر الأمر MOVSB هو أول أمر نتناوله يتعامل مع موقعين في الذاكرة في وقت واحد.

البادئة REP:

يتعامل الأمر MOVSB مع خانة واحدة فقط . ولنقل عدد من الحروف يتم وضع عدد الحروف المطلوب التعامل معها (عدد تكرار تنفيذ الأمر MOVSB) في المسجل CX وبعد ذلك يتم تنفيذ الأمر

```
REP MOVSB
```

وبذلك يتم تنفيذ الأمر MOVSB عدد N من المرات. وتتناقص محتويات CX بعد كل مرة يتم فيها تنفيذ الأمر MOVSB حتى تصبح قيمة CX=0. وبالتالي يمكن كتابة التالي السابق على الصورة

```
CLD
LEASI ,  String1
LEA      DI ,  String2
MOV      CX, 5
REP      MOVSB
```

مثال:

أكتب جزء من برنامج يقوم بنسخ المتغير String1 إلي المتغير String 2 ولكن بصورة معكوسة.

الحل

نجعل المسجل SI يشير إلـي نهايـة المتغيـر الأول (أخر حرف فيه) و DI يشير إلي بدايـة المتغيـر الثاني ونحول الحرف. ثم بعـد ذلـك نـنقص SI (بوضع الرقم 1 في بيرق الاتجاه) ولا ننسى أن نزيد قيمة DI بـ 2 بعد كل مره حيث انه سـيتم إنقـاص محتوياته بمقدار 1 بعد تنفيذ الأمر MOVSB ونحن نريد زيادته بـ 1.

```
        LEA     SI, String1 + 4
        LEA     DI, String2
        STD
        MOV     CX,  5
    MOVE:
        MOVSB
        ADD     DI ,  2
        LOOP    MOVE
```

الأمر MOVSW:

مثل الأمر MOVSB ولكن في هذه الحالة يتم نسخ WORD كاملة بدلاً عن Byte ويكـون المسـجلين

DS: SI يشيران إلي عنوان المصدر والمسجلين ES:DI يشيران إلي المستودع. يتم زيادة أو إنقاص محتويات المسجلين DI, SI بمقدار 2 حسب قيمة بيرق الاتجاه (زيادة في حالة DF = θ ونقصان في حالة أن يكون DF = 1)

مثال:

في المصفوف التالي:

ARR DW 10,20,40,50,60, ?

المطلوب إدخال الرقم 30 وهو يقع بين الرقمين 20 , 40. افترض أن المسجلين DS و ES يشيران إلي مقطع البيانات .

الحل:

يتم نقل الأرقام 50,40 , 60 خانة واحدة وبعد ذلك يمكن إدخال الرقم 30

```
STD
LEA     SI , ARR + 8h     ; SI Points to 60
LEA     DI , ARR +0Ah     ; DI Points to ?
MOV     CX,3
REP     MOVSW
MOV     WORD PTR [DI], 30
```

<u>تخزين نص Storing String:</u>

يستخدم الأمر STOSB لنقل محتويات المسـجل AL في الذاكرة فـي العنـوان المحـدد بالمسـجلين ES:DI . بعد ذلك يتم زيادة محتويات المسـجل DI بواحد إذا كانت DF=0 ويتم إنقاصه إذا كانت DF=1

وبالمثل فان الأمر STOSW يقوم بتخزين محتويات المسجل AX إلي الذاكرة عند العنـوان المحـدد بالمسـجلين ES: DI . ويـتم زيـادة أو نقصـان محتويات المسجل DI حسب قيمة بيرق الاتجاه .

مـثلا لتخـزين الحـرف ' A ' فـي بدايـة المتغيـر String1

```
LEA   DI, String1
MOV     AL, 'A'
CLD
STOSB
```

قراءة وتخزين رسالة نصية:

الخدمة رقم 1 في نداء المقاطعة رقم 21h تقـوم بقراءة حرف واحد فقط . يمكن قـراءة وتخـزين مجموعة من الحروف باستخدام الأمر STOSB .

الإجراء التالي يسمى READ_STR يقـوم بقـراءة مجموعة من الحروف وتخزينها في الذاكرة تنتهي

مجموعة الحــروف بالضـغط علــى مفتـاح الإدخــال Carriage Return .

يتم نداء الإجراء ووضـع عنــوان الإزاحــة للمتغيــر المطلوب قراءة الرسالة بــه فــي المســجل DI يقــوم الإجراء بإعادة عدد الحروف التي تم إدخالها فــي المسجل BX . إذا أخطأ المستخدم في إدخال حرف وضغط على مفتاح الـــ Back_Space يــتم حــذف الحرف من الرسالة وخوارزمية الإجراء هي:

```
Chars_Read = 0
Read a Character
While character is Not a carriage Return Do
    If character is a Back_Space Then
        Chars_Read = Chars_Read - 1
        Remove Previous character from String
    Else
        Store character in String
        Chars_Read = Chars_Read + 1
    End_If
    Read a character
End_While
```

وبلغة التجميع :

```
READ_STR        PROC NEAR
```

```
;READS AND STORES A STRING
;INPUT:        DI ODFFSET OF THE STRING
;OUTPUT:       DI OFFSET OF THE STRING
;              BX=NUMBER OF CHARACTERS
READ
      PUSH    DX
      PUSH    DI
      CLD
      XOR     BX , BX
      MOV     AH , 1
      INT     21H
WHILE1:
      CMP     AL , 0DH
      JE      END_WHILE1
      ;IF CHARACTER IS BACHSPACE
      CMP     AL , 8H

      JNE     ELSE1
      DEC     DI
      DEC     BX
      JMP     READ
ELSE1:
      STOSB
      INC     BX
READ:
      INT     21H
      JMP     WHILE1
```

```
END_WHILE1:
        POP     DI
        POP     AX
        RET
READ_STR        ENDP
```

تحميل نص Load String:

يستخدم الأمر LODSB لتحميل المسجل AL بمحتويات الذاكرة في العنوان المحدد بالمسجلين DS:SI . يتم زيادة أو نقصان المسجل SI بعد تنفيذ الأمر بمقدار 1 وذلك حسب قيمة بيرق الاتجاه .

ويستخدم الأمر LODSW لتحميل المسجل AX بمحتويات الذاكرة في العنوان المحدد بالمسجلين DS:SI . ويتم زيادة أو نقصان المسجل SI بعد تنفيذ الأمر بمقدار 2 وذلك حسب القيمة الموجودة في بيرق الاتجاه .

طباعة نص في الشاشة:

الإجراء التالي المسمي Disp_Str يقوم بطباعة الرسالة يشير إليها المسجل SI عدد الحروف المطلوب طباعتها موجودة في المسجل BX .

```
For count  times Do
```

```
        Load a String Character into Al
        Move  it  to DL
        Output Character
    End_For
```

وهذا هو الإجراء بلغة التجميع

```
    DISP_STR    Proc
    ; inputs   SI : offset of the String
    ;          BX  : No of Characters to Display
    ; Outputs       None
    PUSH        AX
    PUSH        BX
    PUSH        CX
    PUSH        DX
    PUSH        SI
    MOV         CX, BX
    JCXZ        P_EXIT
    CLD
    MOV         AH , 2h
TOP:
    LODSB
    MOV         DL , AL
    INT      21h
    LOOP        TOP

P_EXIT:
    POP         SI
    POP         DX
```

```
    POP         CX
    POP         BX
    POP         AX
    RET
DISP_STR        ENDP
```

البحث في نص Scan String:

يستخدم الأمر SCASB للتأكد من أن الحرف به قيمة محدده هذه القيمة تكون بالمسجل AL . يقوم الأمر يطرح محتويات الذاكرة عند العنوان ES:DI من محتويات المسجل AL وحسب قيمة النتيجة يتم رفع البيارق ولا يتم تخزين النتيجة بعد تنفيذ الأمر. يتم زيادة أو نقصان محتويات المسجل DI حسب قيمة بيرق الاتجاه.

الصورة الثانية للأمر هي SCASW وهى تتعامل مع المسجل AX بدلاً عن AL ولتوضيح الأمر وهى SCSAB أفترض الجزء التالي من البرنامج.

```
String1         DB  'ABC'
:
MOV             AX, @ DATA
MOV             ES, AX
LEA        DI , String1
MOV             AL, 'B'
CLD
SCASB                       ;Scan first byte
```

SCASB ; Scan second Byte

بعد تنفيذ الأمر الأول يكون بيرق الصفر يساوى 0 بحيث أن العملية هي طرح الرقم 41h وهو الحرف 'A' من الرقم 42h وهو الحرف 'B'.

في المرة الثانية سيتم رفع بيرق الصفر وذلك لتساوى القيمتين.

عند البحث عن حرف محدد في نص يتم وضع عدد الحروف المكونة للنص في المسجل CX ويتم تنفيذ الأمر

REPNZ SCASB

حيث يتم طرح كل حرف من محتويات المسجل AX وإنقاص محتويات المسجل CX بواحد حتى يتم العثور علي الحرف المطلوب أو تصل قيمة CX للصفر وذلك عند عدم العثور علي الحرف المطلوب.

مثال:

أكتب برنامج يقوم بحساب عدد الحروف الساكنة Consonants والحروف المتحركة Vowels برسالة.

الحل:

```
    initialize Vowels_Count and
Consonant_Count to zero
    Read and Store a String
    Repeat
        load a String Character
        IF it is a Vowel Then
            Increment Vowel_Count
        else if it is a Consonant Then
            Increment Consonant_Count
        End_IF
    Until End of string
    Display Vowels_Count and
Consonant_Count
```

ويكون البرنامج علي النحو التالي

```
.MODEL SMALL
.STACK 100H
.DATA
   STRING   DB  80 DUP(0)
   VOWELS DB  'AEIOU'
   CONSONANTS     DB
    'BCDFGHJKLMNPQRSTVWXYZ'
   OUT1           DB  0DH,0AH,'VOWELS= $'
   OUT2           DB  ‘CONSONANTS= $'
   VOWELCT        DW 0
   CONSCT DW 0
```

```
.CODE
MAIN PROC
    ; initialize DS
    MOV     AX,@DATA
    MOV     DS,AX
    MOV     ES,AX
    LEA     DX,STRING
    CALL    READ_STR
    MOV     SI,DI
    CLD
REPEAT:
    LODSB
    LEA     DI,VOWELS
    MOV     CX ,5
    REPNE   SCASB
    JNE     CK_CONST
    INC     VOWELCT
    JMP     UNTIL
CK_CONST:
    LEA DI,CONSONANTS
    MOV     CX,21
    REPNE   SCASB
    JNE UNTIL
    INC CONSCT
UNTIL:
    DEC     BX
    JNE     REPEAT
```

```
;OUTPUT NO OF VOWELS
LEA DX,OUT1
MOV     AH ,9
INT 21H
MOV     AX,VOWELCT
CALL    OUTDEC
;OUTPUT NO OF CONSONANTS
LEA     DX,OUT2
MOV     AH ,9
INT     21H
MOV     AX,CONSCT
CALL    OUTDEC
;EXIT TO DOS
MOV     AH,4CH
INT 21H
MAIN        ENDP
INCLUDE PROCFILE.ASM
END     MAIN
```

مقارنة النصوص Compare String:

يستخدم الأمر COPSB لطرح محتويات الذاكرة في العنوان ES:DI من محتويات الذاكرة العنوان DS:SI ويتم تبعاً لذلك رقم البيارق المختلفة ولا يتم تخزين النتيجة . بعد تنفيذ الأمر يتم تحديث محتويات المسجلين SI , DI حسب قيمة بيرق الاتجاه .

الصورة الثانية للأمر هي CMPSW حيث تتعامل مع جمل Words.

```
String1   DB    'ACD'
String2   DB    'ABC'
MOV          Ax, @ DATA
MOV          DS, Ax
MOV          ES, Ax
CLD
LEA       SI, String1
LEA       DI, String2
CMPSB        ;sub 'A' from  'A'
CMPSB        ;sub 'B' from  'B'
CMPSB        ;sub 'C' from  'D'
```

ويتم عادة استخدام التكرار بالأمر REPE (Repeal While equal) عند مقارنة النصوص حيث يتم تكرار عملية المقارنة طالما أن القيمتين متساويتين ولا يتم التوقف إلا إذا لم يتساوى أحد الحرفين أن يكون العداد قد انتهى.

وكمثال افترض أن لدينا متغيرين STR1 و STR2 بطول 10 حروف. المطلوب وضع الرقم صفر في المسجل BX إذا كان النصيين متشابهين ووضع الرقم 1 في المسجل AX إذا كان النص

STR1 ترتيبه قبل النص الثاني ووضع الـرقم 2 إذا كان النص الثاني ترتيبه قبل النص الأول.

```
        MOV     CX,10
        LEASI, STR1
        LEADI, STR2
        CLD
        REPE    CMPSB
        JL   STR1_FIRST
        JG   STR2_FIRST
        MOV     AX, 0
        JMP     EXIT
STR1_FIRST:
        MOV     AX, 1
        JMP     Exit
STR2_FIRST
        MOV     AX,2
EXIT:
```

البحث عن نص فرعي بداخل نص:

هنالك أكثر من طريقة لتحديد أن نـص كبيـر يحتوى على نص صغير بداخلـه مـثلاً إذا أعطينـا التعريف التالي:

```
SUB1    DB  'ABC'
SUB2    DB      'CAB'
MAINST      DB  'ABABCA'
```

لمعرفة أن النص SUB1 موجـود داخـل الـنص الرئيسي يمكن البدء من أول النص حيث

SUB1 ABC
MAINST ABABCA

ولعدم وجود تطابق في الحرف الثالث نحاول ببدء المقارنة من الحرف الثاني

SUB1 ABC
MAINST ABABCA

الحرف الأول غير متطـابق وعليـه ودون مواصـلة المقارنة نرفض هذا الاحتمال ونبداء مـن الحـرف الثالث

SUB1 ABC
MAINST ABABCA

هنا حدث تطابق ويكون SUB1 عبارة عـن نـص صغير SUDSTRING عن النص الكبير وإذا لـم يحدث تطابق تكرر وإذا انتهـى الـنص الكبيـر دون حدوث تطابق كامل يكون النص الصغير غير موجود في النص الكبيـر . ويكـون ذلـك إذا بدأنا عند الحرف المحدد بـ STOP حيث

STOP = MAINST + Length of MAINST - Length of sub string

وهذه هي الخوارزمية

Prompt the use to enter SUBST

```
Read  SUBST
Prompt the User to enter MAINST
READ  MAINST
If(Length of MAINST=0) Or (Length of SUBST=
0) Or SUBST longer than MAINST)
Then
      SUBST Is Not substring of MAINST
Else
      Compute STOP
      Start  = Offset of MAINST
      Repeat
         Compare corresponding chars in
      MAINST (from START on)  and SUBST
            if All chars match then
                  SUBST Found in MAINST
            else
                  START  = START + 1
            END_IF
      Until    (SUBST found in MAINST or
(START > STOP)
END_IF
Display Results
```

الجدول التالي يوضح أوامر التعامل مع النصوص:

الأمر	المستودع	المصدر	صورة	صورة

			الحرف	الكلمة
نسخ	ES:DI	DS:SI	MOVSB	MOVSW
مقارنة	ES:DI	DS:SI	CMPSB	CMPSW
تخزين	ES:DI	AL OR AX	STOSB	STOSW
تحميل	AL OR AX	DS:SI	LODSB	LODSW
بحث (مسح)	ES:DI	AL or AX	SCASB	SCASW

تمارين:

1 – افترض أن المسجل SI به الرقم 100h وان الذاكرة في العنوان 100h بها الرقم 10h

افترض أن المسجل DI به الرقم 00h2 وان الذاكرة في العنوان 101h بها الرقم 15h

افترض أن المسجل AX به الرقم 4142h وان الذاكرة في العنوان 200h بها الرقم 20h

وأن البيرق DF به الرقم 0 وان الذاكرة في العنوان 201h بها الرقم 25h

وضع المصدر والمستودع والقيمة التي يتم التعامل معها في كل من الأوامر التالية ووضع القيمة الجديدة للمسجلين SI , DL

a – MOVSB b- MOVSW c- STOSB
d - STOSW e- LODSB f- LODSW

2. افترض التعريف التالي:

STRING1 DB 'FGHIJ'
STRING2 DB 'ABCDE'
DB 5 DUP (?)

أكتب جزء من برنامج يقوم بوضع النص الأول في نهاية النص الثاني لإصدار النص ABCDEFGHIZ

3. أكتب جزء من برنامج يقوم بتبديل النصين في المثال السابق

4. نص يتضمن بالحرف الذيل كوده 0 مثل

STR DB 'this is an ASCIIz String' , θ

اكتب إجراء يسمى Length يستقبل عنوان الإزاحة للنص المسجل DX ويقوم بإرجاع طول النص في المسجل CX .

5. باستخدام أنماط العنونة المختلفة اكتب مجموعة من الأوامر تقوم بتنفيذ كل من التالي:

a - MOVSB b- STOSB c- LODSB
d- SCASB e- CMPSB

6. افترض التعريف التالي:

String DB ' TH *S* AR'

قم بكتابة برنامج يقوم بطباعة الرسالة السابقة بعد استبدال الحرف '*' بالحرف 'E'

7. افترض التعريف التالي:

String1 DB 'TH I S I S A T E S T'
String2 DB 11 DUP (?)

اكتب جزء من برنامج يقوم بنسخ النص الأول إلي الثاني بعد إزالة المسافات من النص.

برامج للكتابة:

8. هنالك مجموعة من الجمل التي تقرأ من الاتجاهين لتعطى نفس الجملة مثل ' MADAM I AM ADAM " ويتم استبعاد المسافات والعلاقات الخاصة من الجملة.

أكتب برنامج يقوم بقراءة نص، ثم طباعته من الأمام ومن الخلف (معكوس) في سطرين متتاليين . بعد ذلك يقوم بتحديد هل النص من النوع الذي يمكن قراءته من الاتجاهين.

9. في الجداول يتم عادة طباعة الأرقام بمحاذاة لجهة اليمين مثل:

```
  123
12465
  131
```

المطلوب كتابة برنامج يقوم بقراءة عشرة أرقام الواحد بطول يصل حتى 10 خانات. ثم طباعة هذه الأرقام بالشكل المطلوب

10. اكتب برنامج يقوم بقراءة نصين وتحديد أيهما يأتي أبجديا قبل التالي

11. اكتب إجراء يسمى INSERT والذي يقوم بإدخال النص STRING1 داخل النص الثاني STRING2 في مكان محدد.

المدخلات: SI يحتوى على عنوان الإزاحة للنص الأول

DI يحتوى على عنوان الإزاحة للنص الثاني

BX يحتوى على طول النص الأول

CX يحتوى على طول النص الثاني

AX يحتوى على عنوان الإزاحة المطلوب إدخال النص فيه

المخرجات: DI يحتوى على عنوان الإزاحة للرسالة الجديدة

BX يحتوى على طول النص الجديد

اكتب برنامج يقوم بقراءة نصين ورقم صحيح N ونداء الإجراء INSERT وبعد ذلك طباعة النص الجديد

12. اكتب إجراء يسمى DELETE والذي يقوم بحذف N حرف من نص من مكان محدد وملئ الفراغ الناتج من ذلك.

المدخلات: DI يحتوى على عنوان الإزاحة للنص

BX طول النص

CX عدد الحروف المطلوب مسحها

SI عنوان الإزاحة للمكان المطلوب الحذف ابتداء منه

المخرجات: DI عنوان الإزاحة للنص الجديد

BX طول النص الجديد

أكتب برنامج يقوم بقراءة النص والحرف المطلوب المسح منه وعدد الحروف المطلوب مسحها. ثم نداء الإجراء DELETE ثم طباعة النص الجديد.

الفصل الحادي عشر
تطبيقات عملية
Practical Applications

في هذا الفصل سنتناول بعض الأمثلة العملية والتي تستخدم فيها لغة التجميع لأداء بعض المهام، في أغلب هذه التطبيقات نقوم باستخدام الخدمات التي يقدمها نظام التشغيل في تنفيذ بعض المهام

التطبيق الأول : معرفة إصدارة نظام التشغيل التي يعمل في النظام

في هذا التطبيق يتم استخدام الخدمة رقم 30h لنداء المقاطعة Int 21h والتي تحدد رقم إصدارة نظام التشغيل وهي عبارة عن الرقم الصحيح للإصدارة ورقم كسري مثل 6.22 والذي يعني أن إصدارة نظام التشغيل هي القيمة الأساسية Minor تساوي 6 والقيمة الصغرى 22 وهكذا، بعد هذا النداء يتم الاحتفاظ بهذه القيم والتي تقوم تلك الخدمة بتجهيزها في المسجلين AL و AH في متغيرين في الذاكرة ليتم طباعتهما لاحقاً.

;=====================================

```
==================================
;    program: DosVer.asm
;    purpose: gets the DOS Version  using
;interrupt 21h function 30h

;    purpose: gets the DOS Version  using
interrupt 21h function 30h
;    input  : None
;    output : Minor and Major versions
;    usage  : OUTDEC procedure in procfile.asm
;    update :
;===========================================
==================================
.MODEL SMALL
.STACK 100H
.DATA
     CR  EQU 0DH
     LF   EQU 0AH
     MAJOR DB '?'
     MINOR  DB '?'
     MSG   DB 'GET DOS VERSION:INT 21H
     FUNCTION 3OH',CR,LF,'MS-DOS
               Version ','$'

     MSG1  DB CR,LF,'MAJOR VERSION
NUMBER IS :$'
     MSG2  DB CR,LF,'MINOR VERSION
```

```
NUMBER IS :$'
.CODE
MAIN PROC
    ;initialization
    MOV AX,@DATA
    MOV DS,AX
    ;get dos version
    MOV AH,30H
    INT 21H
    MOV MAJOR,AL
    MOV MINOR ,AH
    ;display results
    LEA DX,MSG
    MOV AH,9h
    INT 21H
    LEA DX,MSG1
    MOV AH,9h
    INT 21H
    XOR AX,AX
    MOV AL,MAJOR
    CALL OUTDEC
    LEA DX,MSG2
    MOV AH,9h
    INT 21H
    XOR AX,AX
    MOV AL,MINOR
    CALL OUTDEC
```

```
        ;return to dos
        MOV AH,4CH
        INT 21H
MAIN    ENDP
Include Procfile.asm
END     MAIN
```

التطبيق الثاني : معرفة تاريخ اليوم

في هذا التطبيق يتم استخدام الخدمة رقم 2Ah لنداء المقاطعة Int 21h والتي يتم فيها معرفة تاريخ اليوم من النظام كما هو موضح في الجزء التالي :

```
;=====================================
===============================
;    program: sysDate.asm
;    purpose: gets the year,month,day,and day of the
week
;            from the system using interrupt 21h
function 2Ah
;    Calling Registers  : AH = 2A
;    Return registers:
;      CX : year(1980 - 2099)
;      DH : month(1 - 12)
;      DL : day(1 - 31)
;      AL : day of the week (0 =Sunday, 1
=Monday,etc )
;    usage  : OUTDEC procedure in procfile.asm
```

```
;    update : 27/11/2000
;========================================
================================
.MODEL       SMALL
.STACK 100H
.DATA
      CR  EQU      0DH
      LF         EQU      0AH
      MSG       DB  'GET SYSTEM DATE :INT 21H
FUNCTION 2A',CR,LF
      DB  'YEAR  :$'
      YEAR     DW        '?'
      MSG2     DB  CR,LF,'MONTH :$'
      MONTH DB  '?'
      MSG3     DB  CR,LF,'DAY   :$'
      DAY       DB  '?'
      MSG4     DB  CR,LF,'DAY OF WEEK:','$'
      Dweek    DB  '?'
      SUN       DB  'Sunday $'
      MON       DB  'Monday $'
      TUES      DB  'Tuesday $'
      WEDN     DB  'Wednesday $'
      THURS  DB  'Thursday $'
      FRID      DB  'Friday $'
      SATDB  'Saturday $'
.CODE
MAIN PROC
```

```
;initialization
 MOV     AX,@DATA
 MOV     DS,AX
 ;get system date
 MOV     AH,2AH
 INT 21H
 ;assign values of date
 MOV     YEAR,CX
 MOV     MONTH,DH
 MOV     DAY,DL
 MOV     Dweek,AL
 ;
 MOV     DL,dWEEK
 MOV     AL,2H
 INT 21H
 ;display values of date
 LEA     DX,MSG
 MOV     AH,09H
 INT 21H
 ;year
 MOV     AX,CX
 CALL    OUTDEC
 ;month
 LEA     DX,MSG2
 MOV     AH,09H
 INT 21H
 XOR     AX,AX  ;clear AH and AL
```

```
MOV     AL,MONTH
CALL    OUTDEC
;day
LEA     DX,MSG3
MOV     AH,09H
INT 21H
XOR     AX,AX
MOV     AL,DAY
CALL    OUTDEC
; display the equivalent day of week
LEA     DX,MSG4
MOV     AH,09H
INT 21H
CMP     Dweek,0
JE   ZERO
CMP     Dweek,1
JE   ONE
CMP     Dweek,2
JE   TWO
CMP     Dweek,3
JE   THREE
CMP     Dweek,4
JE   FOUR
CMP     Dweek,5
JE   FIVE
CMP     Dweek,6
JE   SIX
```

```
    JMP     END_CASE
ZERO:
    LEA     DX,SUN
    JMP     DISPLAY_
ONE:
    LEA     DX,MON
    JMP     DISPLAY_
TWO:
    LEA     DX,TUES
    JMP     DISPLAY_
THREE:
    LEA     DX,WEDN
    JMP     DISPLAY_
FOUR:
    LEA     DX,THURS
    JMP     DISPLAY_
FIVE:
    LEA     DX,FRID
    JMP     DISPLAY_
SIX:
    LEA     DX,SAT
DISPLAY_:
    MOV     AH,09H
    INT 21H
END_CASE:
    MOV     AH,4CH
    INT 21H
```

```
MAIN ENDP
Include   procfile.asm
END       MAIN
```

التطبيق الثالث : معرفة الزمن

في هذا التطبيق يتم استخدام الخدمـة رقـم 2Ch لنـداء المقاطعة Int 21h والتي يتم عن طريقها معرفة الزمن مـن الساعة الموجودة في النظام وذلك على النحو التالي:

```
;====================================
==============================
;    program: sysTime.asm
;    purpose: gets the hour,minutes,seconds,and
hundredth  of seconds
;              from the system using
;    calling registers:  AH = 2Ch
;    return  registers:  CH =Hour(O - 23)
;                        CL =Minutes(O - 59)
;                        DH =Seconds(O - 59)
;                        DL =Hundredths of seconds(O - 99)

;    input  : None
;    output : hour,minutes,seconds,and hundredth of
seconds
;    usage  : OUTDEC procedure in procfile.asm
;    update : 28/11/2000
```

```
;=====================================
==============================
.MODEL SMALL
.STACK 100H
.DATA
      CR     EQU 0DH
      LF     EQU 0AH
      MSG    DB  'GET SYSTEM TIME :INT 21H
FUNCTION 2C',CR,LF,'$'
      TM     DB  ?
.CODE
MAIN PROC
      ;initialization
      MOV      AX,@DATA
      MOV      DS,AX
      ;print msg
      LEA      DX,MSG
      MOV      AH,09H
      INT 21H
      ;get system time
      MOV      AH,2cH
      INT 21H
      ;assign values of time
      MOV      BX,DX    ; store sec and hundred of
secs from DX
      XOR      AX,AX    ; ax:=zero
      MOV      AL,CH    ;hour
```

```
    CMP     AL,12d
    JG  GREAT
    MOV     TM,'a'
    jmp     CONTINUE
GREAT:
    SUB     AL,12
    MOV     TM,'p'
CONTINUE:
    CALL    OUTDEC
    MOV     DL,':'
    MOV     Ah,02H
    INT 21H
    AND     AX,0    ;ax:=zero
    MOV     AL,CL   ;minutes
    CALL    OUTDEC
    MOV     DL,':'
    MOV     Ah,02H
    INT 21H
    MOV     AX,0    ;ax:=zero
    MOV     AL,BH   ;seconds
    CALL    OUTDEC
    MOV     DL,'.'
    MOV     Ah,02H
    INT 21H
    MOV     AX,0    ;  ax:=zero
    MOV     AL,Bl   ;hundred of seconds
    CALL    OUTDEC
```

```
        ;print space
        MOV     DL,' '
        MOV     AH,02H
        INT 21H
        MOV     DL,TM
        MOV     AH,02H
        INT 21H
        ;return to dos
        MOV     AH,4CH
        INT 21H
MAIN    ENDP
Include ProcFile.asm
END     MAIN
```

التطبيق الرابع : تغيير التاريخ

في هذا التطبيق يتم استخدام الخدمة رقم 2Bh لنداء المقاطعة Int 21h والتي يتم عن طريقها تغيير الزمن للنظام وذلك على النحو التالي :

```
TITLE Setdate.asm
;=====================================
==================================
;    Purpose: sets the System date using interrupt
21h
;    function 2Bh
;    Calling Registers :
```

```
;                   AH = 2B H
;                   CX : year(1980 - 2099)
;                   DH : month(1 - 12)
;                   DL : day(1 - 31)
;    Return Registers :
;                   AL = 00 if success to change the
system date
;    usage  : INUNDEC procedure in procfile.asm
;    update : 27/11/2000
;=======================================
==============================
.MODEL SMALL
.STACK 100H
.DATA
      LF EQU 0DH
      CR EQU 0AH
      prompt   DB        LF,CR,'Enter The Day   : $'
      MSG_M        DB         LF,CR,'Enter The Month
: $'
      MSG_Y        DB  LF,CR,'Enter The
Year(1980..2099) : $'
      MSGSUC       DB  LF,CR,'Your Date Is
Changed.$'
      MSGFAIL      DB  LF,CR,'Your Date Is Not
Changed.'
                   DB  LF,CR,'Do You Want To Try
Again Y/N? $'
```

```
    MSGINV      DB  LF,CR,'Invalid Date...'
                DB  LF,CR,'Do You Want To Try
Again Y/N? $'
    year        DW '?'
    month       DB  '?'
    day         DB  '?'
.CODE
MAIN PROC
    MOV     AX,@DATA
    MOV     DS,AX
begin :
    ; Display Prompy Message
    MOV     AH,9
    LEA     DX , prompt
    INT 21H
    ;  Read the Day
    CALL    INUNDEC
    CMP     AL , 1
    JL   begin
    CMP     AL , 31D
    JG   begin
    MOV     DAY , AL
@month :
    MOV     AH , 9
    LEA     DX , MSG_M
    INT 21H
    ;  Read the Month
```

```
    CALL    INUNDEC
    CMP     AL , 1
    JL   @MONTH
    CMP     AL , 31D
    JG   @MONTH
    ;CALL INUNDEC
    MOV     MONTH , AL
@YEAR :
    MOV     AH,9
    LEA     DX , MSG_Y
    INT 21H
    ;  Read the Year
    CALL    INUNDEC
    CMP     AX , 1980D
    JL   @YEAR
    CMP     CX , 2099D
    JG   @YEAR
    ; Set Date using Function 2Bh
    MOV     CX , AX         ; CX = The Year
    MOV     DH , MONTH        ; DH = The Month
    MOV     DL , DAY          ; DL = The Day
    MOV     AH , 2BH
    INT 21H
    ;IS DATE CHANGED ?
    CMP     AL , 00H
    JNE     AGAIN
    MOV     AH , 9H
```

```
    LEA     DX , MSGSUC
    INT 21H
    JMP     EXIT
again:
    MOV     AH , 9H
    LEA     DX , MSGFAIL
    INT 21H
answer:  ;ANSWER Y/N
    MOV     AH , 1H
    INT 21H
    CMP     AL , 'Y'
    JE  begin
    CMP     AL , 'y'
    JE   begin
    CMP     AL , 'n'
    JE   EXIT
    CMP     AL , 'N'
    JE   EXIT
    JMP     ANSWER
exit:
    MOV     AH , 4CH
    INT 21H
MAIN    ENDP
include procfile.asm
END     MAIN
```

التطبيق الخامس : تغيير الزمن

في هذا التطبيق يتم استخدام الخدمة رقم 2Dh لنداء المقاطعة Int 21h والتي يتم فيها تغيير الزمن في ساعة النظام وذلك على النحو التالي :

```
TITLE Settime.asm
;======================================
==================================
;    Purpose: sets the System time using interrupt
21h
;    function 2Dh
;    Calling Registers :
;                    AH = 2D H
;                    CH : Hours (0..23)
;                    CL : Minutes (0..59)
;                    DH : Seconds (0..59)
;    Return Registers :
;                   AL = 00 if success to change the
system time
;    usage  : INUNDEC procedure in procfile.asm
;    update : 27/11/2000
;======================================
==================================
.MODEL SMALL
.STACK 100H
.DATA
  LF EQU 0DH
```

```
  CR EQU 0AH
  PROMPT  DB       LF,CR,'Enter The Hour(0..23) :
$'
  MSG_M   DB       LF,CR,'Enter The Minute(0..59)
: $'
  MSG_S   DB       LF,CR,'Enter The Second(0..59)
: $'
  MSGSUC  DB       LF,CR,'Your time is changed.$'
  MSGFAIL DB       LF,CR,'Your Time Is Not
Changed.'
          DB       LF,CR,'Do You Want To Try Again
Y/N? $'
  MSGINV  DB       LF,CR,'Invalid Time...'
          DB       LF,CR,'Do You Want To Try Again
Y/N? $'
  HOUR    DB       '?'
  MINUTE  DB       '?'
.CODE
MAIN  PROC
      MOV      AX,@DATA
      MOV      DS,AX
begin :
      ; DISPLAY PROMPT MESSAGE
      MOV      AH , 9
      LEA      DX , prompt
      INT 21H
      ; Read The Hour
```

```
    CALL    INUNDEC
    MOV     HOUR , AL
    CMP     AL , 23D
    JG  begin
@minute:
    MOV     AH , 9
    LEA     DX , MSG_M
    INT 21H
    ; Read the Minute
    CALL    INUNDEC
    CMP     AL , 59D
    JG  @minute
    MOV     MINUTE , AL
@second :
    MOV     AH,9
    LEA     DX , MSG_S
    INT 21H
    ; Read The Second
    CALL    INUNDEC
    CMP     AL , 59D
    JG  @second
    ; Set Time using Function 2Dh
    MOV     DH , AL         ; DH = Seconds
    MOV     CL , MINUTE       ; CL = Minutes
    MOV     CH , HOUR        ; CH = Hour
    MOV     AH , 2DH
    INT 21H
```

```
        ;IS DATE CHANGED ?
        CMP     AL , 00H
        JNE     AGAIN
        MOV     AH , 9H
        LEA     DX , MSGSUC
        INT 21H
        JMP     EXIT
again:
        MOV     AH , 9H
        LEA     DX , MSGFAIL
        INT 21H
answer:  ;ANSWER Y/N
        MOV     AH , 1H
        INT 21H
        CMP     AL , 'Y'
        JE   begin
        CMP     AL , 'y'
        JE   begin
        CMP     AL , 'n'
        JE   EXIT
        CMP     AL , 'N'
        JE   EXIT
        JMP     ANSWER
exit:
        MOV     AH , 4CH
        INT 21H
MAIN    ENDP
```

```
include procfile.asm
END       MAIN
```

التطبيق السادس : مقارنة بين لغات البرمجة العالية والبرمجة بلغة التجميع

في هذا التطبيق المطلوب كتابة حروف علي الشاشة، معلوم أن الشاشة يمكن الكتابة فيها مباشرة وذلك عن طريق الكتابة في المنطقة الخاصة بها في الذاكرة (وهي في حالة كروت الشاشة من النوع SVGA والمستخدمة في الجامعة تبدأ من العنوان الفيزيائي B8000h) حيث يتم كتابة الكود الـ ASCII للحرف متبوعاً بخصائص الحرف Attribute وهي عبارة عن لون الحرف ولون الخلفية التي سيتم طباعته عليها.

وسيتم ملئ الشاشة بحروف لمقارنة سرعة البرامج المكتوبة بلغة التجميع والبرامج المكتوبة بإحدى اللغات الأخرى مثل لغة الباسكال، نسبة للسرعة العالية لبرنامج لغة التجميع سيتم في هذه المقارنة استخدام برنامج يقوم بملء الشاشة بالحروف من A إلي Z (في كل مرة يتم ملء الشاشة بالحرف المحدد) ويتم تكرار هذه العملية عدد 9 مرات وذلك لأننا سنقوم بمعرفة الزمن قبل البدء في البرنامج

ومعرفة الزمن بعد الانتهاء من التنفيذ وإيجاد الزمن الذي استغرقه البرنامج في التنفيذ.

الطريقة الأولي : باستخدام لغة الباسكال والعبارة Write :

```
program    displayrun;
   uses crt,Dos;
  var
    hs, ms, ss, hunds,he, me, se, hunde : Word;
    ch:char;
    BX, Counter:integer;
begin
  clrscr;
  TextColor(blue);
  TextBackground(white);
  GetTime(hs,ms,ss,hunds);
  FOR BX:= 1 TO 9 DO
   for ch:='A' to 'Z' do
     for counter :=1 to 2000 do
       write(ch);
  GetTime(he,me,se,hunde);
  writeln;
  writeln('Started  at  ',hs,':',ms,':',ss,'.',hunds);
  writeln('Finished at  ',he,':',me,':',se,'.',hunde);
  writeln('Run time is  ',he-hs,':',me-ms,':',se-
ss,'.',hunde-hunds);
  repeat  until keypressed;
```

```
end.
```

الطريقة الثانية : باستخدام لغة الباسكال والعبارة والتعامل مع الذاكرة مباشرة:

```
program    displayrun;
   uses crt,Dos;
  var
    hs, ms, ss, hunds,he, me, se, hunde : Word;
    ATRIB,ch:BYTE;
    BX, Counter:integer;
begin
  clrscr;
  TextColor(blue);
  TextBackground(white);
  GetTime(hs,ms,ss,hunds);
  ATRIB:=$17;
  FOR BX:= 1 TO 9 DO
   for ch:=65 to 90 do
     for counter :=0 to 2000 do
      BEGIN
       MEM[$B800:2*COUNTER]:=CH;
       MEM[$B800:2*COUNTER+1]:=ATRIB;
      END;
{       write(ch);}
  GetTime(he,me,se,hunde);
  writeln;
```

```
  writeln('Started  at  ',hs,':',ms,':',ss,'.',hunds);
  writeln('Finished at  ',he,':',me,':',se,'.',hunde);
  writeln('Run time is  ',he-hs,':',me-ms,':',se-
ss,'.',hunde-hunds);
end.
```

الطريقة الثالثة : باستخدام لغة التجميع :

```
TiTle Disp_asm : Fill The screen & Compute
Runtime

.MODEL SMALL

.STACK 100H

.DATA

     printCh   dw '?'
     MSGS    DB  0DH,0AH,'Start  Time is  $'
     Hs   DB  '?'
     Ms   DB  '?'
     Scs DB  '?'
     HSs        DB  '?'
     MSGe    DB  0DH,0AH,'Finish Time is  $'
     He   DB  '?'
     Me   DB  '?'
     Se   DB  '?'
```

```
        HSe       DB  '?'
        MSGR    DB  0DH,0AH,'Run    Time is  $'
.CODE
MAIN PROC
        ;initialization
        MOV      AX , @DATA
        MOV      DS , AX
        ; Get start time
        MOV      AH,2CH
        INT        21H
        MOV      Hs , CH
        MOV      Ms , CL
        MOV      Scs , DH
        MOV      HSs , DL
        MOV      AX,0B800h   ;color active display
page
        MOV      DS,AX
        MOV      AH,17H
        MOV      BX,9
DISPLAY_ALL:
        MOV      AL,41h
AGAIN:
MOV      DI,0
        MOV      CX,2000d
        ;fill active display page
        FILL_BUF:
MOV      [DI],AX
```

```
ADD      DI,2
LOOP     FILL_BUF     ;loop until done
    ADD      AX,01H
    CMP      AL,'Z'
    JLE      AGAIN
    DEC      BX
    JNZ      DISPLAY_ALL
    ; Get finish time
    MOV      AX , @DATA
    MOV      DS , AX
    MOV      AH,2CH
    INT      21H
    MOV      He , CH
    MOV      Me , CL
    MOV      Se , DH
    MOV      HSe , DL
        ; display start time
MOV      AH , 9
    LEA      DX , MSGs
    INT 21H
    XOR      AX , AX
MOV      AL , Hs
CALL     OUTDEC
MOV      DL , ':'
    MOV      AH , 2
    INT 21H
    ;
```

```
XOR     AX , AX
    MOV     AL , Ms
    CALL    OUTDEC
    MOV     DL , ':'
    MOV     AH , 2
    INT 21H
    ;
XOR     AX , AX
    MOV     AL , Scs
    CALL    OUTDEC
    MOV     DL , '.'
    MOV     AH , 2
    INT 21H
    ;
    XOR     AX , AX
    MOV     AL , HSs
    CALL    OUTDEC
    MOV     DL , ':'
    MOV     AH , 2
    INT 21H
    ; display finish time
    MOV     AH , 9
    LEA     DX , MSGe
    INT 21H
    XOR     AX , AX
    MOV     AL , He
    CALL    OUTDEC
```

```
    MOV     DL , ':'
    MOV     AH , 2
    INT 21H
    ;
    XOR     AX , AX
    MOV     AL , Me
    CALL    OUTDEC
    MOV     DL , ':'
    MOV     AH , 2
    INT 21H
    ;
    XOR     AX , AX
    MOV     AL , Se
    CALL    OUTDEC
    MOV     DL , '.'
    MOV     AH , 2
    INT 21H
    ;
    XOR     AX , AX
    MOV     AL , Hse
    CALL    OUTDEC
    MOV     DL , ':'
    MOV     AH , 2
    INT 21H
    ; display run time
MOV     AH , 9
LEA     DX , MSGR
```

```
    INT 21H
    XOR     AX , AX
    MOV     AL , He
    SUB     AL , Hs
CALL    OUTDEC
MOV     DL , ':'
    MOV     AH , 2
    INT 21H
    XOR     AX , AX
    MOV     AL , Me
    SUB     AL , Ms
    CALL    OUTDEC
    MOV     DL , ':'
    MOV     AH , 2
    INT 21H
    XOR     AX , AX
    MOV     AL , Se
    SUB     AL , Scs
    CALL    OUTDEC
    MOV     DL , '.'
    MOV     AH , 2
        INT 21H
    XOR     AX , AX
    MOV     AL , HSe
    SUB     AL , HSs
    CALL    OUTDEC
; dos exit
```

```
        MOV     AH,4CH
        INT     21H
MAIN    ENDP
      Include procfile.asm
END  MAIN
```

المقارنة:

بعد تشغيل البرامج الموضحة أعلاه ومقارنة زمن التنفيذ لكل منها. ما هو البرنامج الذي استغرق أقل زمن في التنفيذ؟ وما هو تعليقك على ذلك؟

محمد إسماعيل محمد

becasod@hotmail.com

becaso_d@yahoo.com

moonweep@hotmail.com